Ｖｅｎｓｉｍ６モデリングガイド

日本語訳ドラフト　Version.2

日本未来センター

システムダイナミクスグループ

福島 史郎 訳

2014年1月

はしがき

日本未来研究センターは、２００２年４月に設立されたＮＰＯ法人で、以下の３つのミッション（使命）を遂行し、日本および世界の未来の持続可能な発展（Sustainable Development）に貢献すべく、微力ならが１０年以上にわたり活動に取り組んできました。

・システムダイナミックスの普及

・未来ビジョンが交換・学習できる交流の場（フォーラム）の提供

・持続可能な地球モデル村および未来博物館の建設・運営

すなわち、未来をコンピュータ・シミュレーションの手法を用いてデザインし、コンピュータではデザインできない場合には私たちの頭脳を用いてデザインし、それらに立脚しつつ持続可能な地球村、社会を建設してゆきたいというのが私たちのミッションです。

その最初の普及ミッションであるシステムダイナミックス(System Dynamics）とは、ストック、フロー、変数、矢印の４つのアイコンを組み合わせることにより、企業の経営戦略、政府・自治体の公共政策、地球環境問題等々多岐にわたる複雑なシステムをお絵かき感覚でモデリングし、その構造をシミュレーション分析する手法で、１９５０年代にＭＩＴ（マサチューセッツ工科大学）のジェイ・フォレスター教授によって開発されたものです。４つのアイコンを組み合わせてお絵かきをするといった簡単なモデリング手法のために、

米国ではK−１２(初等・中等) 教育にもシステムダイナミックスが普及しています。

私たちは、システムダイナミックスの代表的ソフトである Vensim の国内販売をさせていただく機会をえ、これまでその日本語化にも取り組んできました。今回、同センターのシステムダイナミックス研究員である福島史郎さんの尽力により、長年の念願であった中級レベルの Vensim 6 Modeling Guide の日本語訳が完成しました。ユーザーズガイドでモデリングの基礎を習得された皆さんが、より高度で広範な分野のアプリケーションモデルを構築する際に不可欠となる手法が習得できる指南書です。をこれを機会にシステムダイナミックスによるシステム構造分析、未来システム思考が広く普及し、日本の社会がより持続可能な社会に進化してゆくことを念願しています。

NPO 法人日本未来研究センター

理事長　山口　薫,　Ph.D.

２０１３年１２月

翻訳にあたって

私は2004年4月、電機メーカーの開発生産性向上をミッションとするスタッフ部門の責任者として、同志社ビジネススクールの門を叩いた。開発現場において同じ様な失敗が繰り返される問題を、山口薫先生の指導でシステムダイナミックスを用いて解明する中で、システムダイナミックスがパワフルなツールであることを学んだ。山口薫先生のゼミでは、Vensimのモデリングガイドで解説されている挙動を履修生に再現させ、それを生み出すシステム構造を説明させるという演習が、３ヶ月にわたり毎週課された。今にして思えば、システムダイナミックスによるビジネスモデリングのテクニックが一番身についたと思える３ヶ月であった。システムダイナミックスは4種類のアイコンを使ってシステム構造をモデリングし、シミュレーションを繰り返しながら問題解決してゆく手法であるが、それはとてもシンプルであり、極端に言えば小中学生でも使うことができる。そういうこともあってか、システムダイナミックスをビジネス上の課題に適用するためのモデリングテクニックを学ぶ入門のテキストがなかなか見当たらないのが実情である。小中学生でも使うことができるシステムダイナミックスであるが、ビジネスに適用するとなると、開発、生産、販売、市場、モチベーションそして財務等といったビジネスに関するシステム構造をモデリングするためのベーシックなテクニックは、一通り学習した方が良いと思われる。つまり、他人の知恵を活用できるところは活用して時間を節約して、問題の本質を考えるための時間を十分に確保するべきである。このモデリングガイドには、ビジネスモデリングにおいて頻度高く活用できそうな要

素が数多くちりばめられている。また、システムダイナミックスのモデリングには特徴的ないくつかのアプローチがあるが、このモデリングガイドではあえて、異なったアプローチでモデリングして、モデリングアプローチの違いを例示している。システムダイナミックスのモデリングアプローチのバリエーションを効率よく学ぶという意味でもこのモデリングガイドは利用価値がある。このように、入門テキストとしても有用なモデリングガイドを、システムダイナミックスによるビジネスモデリングを志すより多くの人々に手軽に活用戴くために日本語訳した。ただ、残念ながら小生の理解不足から、誤訳や不十分な表現も残されていると思うが、そこは平にご容赦願いたい。このモデリングガイドが、システムダイナミックスによるビジネスモデリングを志す人材の養成の一助となれば喜びとするところである。

最後になりましたが、親身にご指導戴いた山口薫教授に感謝の意を表したいと思います。また、山口薫教授のもとでともにビジネスモデリングを研究した大学院生の佐藤安弘氏からは、ディスカッションを通して、ビジネスの現場で理解しやすい変数名の日本語訳のアイディアを戴きました。また、本原稿をレビュー戴いたNPO法人日本未来研究センターの樽本祐介研究員ならびに出版をご手配戴いた同センターの出口亘大阪代表にお礼申し上げたいと思います。

NPO 法人日本未来研究センター

システムダイナミクスグループ 研究員

経営学博士（ＳＤ専攻） 福島 史郎

２０１３年１２月

目次

第１章
システムダイナミクスの概観

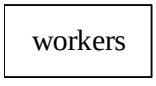
1.1 はじめに

このモデリングガイドでは、SDモデルを構築・使用するための基本概念を紹介しています。理解を深めるために、できるだけ多くの例を挙げて説明する様にしています。まだVensimに慣れていないのであれば、ユーザーズガイドを先にご覧になってモデルを作成することをお薦めします。

このガイドでは、各章でモデルが紹介されていますが、それぞれの章を読み進みながら、順を追ってモデル作成作業を行えば、モデルを完成させることができる様になっています。このモデリングガイドで紹介したモデルは、Vensimをインストールしたコンピュータの所定のディレクトリ内にあります。[1] 完成モデルが保存されているサブディレクトリ名は、その章番号から始まり、ガイドで用いられているタイトルかサブタイトルをファイル名として用いています。モデルを自ら作成するときは、Vensimのサンプルが収められているディレクトリとは別の場所に名前を変えて保管する様にしてください。このガイドではモデルを構築するための細かい操作方法は記載していません。もし、操作が上手く行かない場合はユーザーズマニュアルに戻ることをお薦めします。

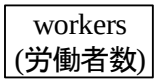
この和訳版では、モデルを図示する場合には、原則として変数名を英文と括弧つきの和訳を併記する形で表示しています。

英文表記



attrition

和訳の併記



attrition
(人員減)

なお、式を表示する場合は和訳を括弧つきで併記すると、関数や表変数の引数と混同して理解しにくくなるので、式の表示については英文でのみ表記しています。

[1] Vensim のヘルプメニューからモデリングガイドを参照すれば、その冒頭にモデルが保存されているディレクトリへのリンクが張られています。モデルは英語で表記されています。

1.2 事象・振る舞いと構造(Events, Behavior and Structure)

人生を考えてみれば、SDで言うところの事象(Events)に相当するものをたくさん見出すことができます。例えば、誕生パーティー、卒業式、就職、新製品の発売、引退、議論、契約そして嵐といったことがことごとく事象に相当します。事象は誰もが良くそれを理解しているので、議論がそのことに終始してしまう傾向があります。しかしながら、自分達の住んでいる世界を理解するという意味では、事象はあまり役に立たないということが分かります。事象が自分達の世界の理解に役に立たないということは、物理学の話をすれば良く分かります。　教師が生徒の前に立って、「チョークはなぜ落ちたのか?」と尋ねたとしましょう。生徒は、「先生がチョークを手から放したからです」と答えクスクスと笑って話は終わってしまいます。同じ学生にスーツを着せて「なぜ株価が下落したのですか?」と尋ねた場合でも、「連邦準備制度理事会が金利を上げると発表したからです」とまじめに答えるでしょう。この例の様に、事象に着目した議論だけに終始すれば、その背後にある物事の本質に迫ることは不可能だということが分かります。

事象から一歩引いて考えるということは、「振る舞い(Behavior)のパターン」に着目するということになります。　振る舞いのパターンとは、長期間にわたって発生した一連の事象のことです。例えば、アメリカ革命は1つの事象ですが、当時の人権抑圧の程度、民衆の憤りの程度や革命に先だつ数十年間の税負担の程度といったものは、振る舞いのパターンと考えられます。　事象から一歩遠ざかって、「何がその事象を引き起こしたのか?」という問いを考えることで振る舞いのパターンについて考えることができる様になります。そうすれば、個々の事象の理解に留まることなく、その背景にある、はるかに深い意味にまでたどり着くことができるのです。ここまでたどり着くことができれば、「この事象の後にどの様な事象が続くのであろうか?」といった個々の事象に関する興味は小さくなってきます。そのかわり、物事を変化させる圧力や不均衡の源に興味が向かうことになります。

構造(Structure)とは、振る舞いを引き起こす物理的な相互結合や情報の相互連結の集まりです。「在庫とは生産と出荷の差が累積したものである」というのが構造です。また、「労働者数は採用と退職の人員数で変化する」というのが構造です。そして、「需要と適正在庫の差を埋めるために設定された生産目標に基づいて人員を採用する」というのも構造です。この構造の結果として「在庫水準が変動する」というのが「振る舞い」であり、「在庫を持ちすぎて利益が上がらなくなり、CEOがクビになった」ということが事象である、というができます。構造は振る舞いを決定します。そして事象は振る舞いのスナップショット写真の様なものであるといえます。

事象、振る舞い、構造を区別することは、問題を理解し対応するために大変重要です。つまり、上手に政策や調整を行うためには、構造を変える必要があるのです。構造を変えれば振る舞いは改善されます。構造を良い方向に変化させれば、仮に好ましくない象が起こったとしても、変化させる前に比べて頻繁が下がります。システムダイナミックスとVensimは、構造を表現し、構造がどの様に振る舞いを決定するかを理解するツールなのです。

1.3　システムダイナミックスプロセス

良いSDモデルを開発するための普遍的に使えるプロセスといったものはありませんが、実践的に有効な基本的手段といえるものはあります。モデルコンセプトガイドライン[2]には、モデリングプロセスについて述べられています。その中から役に立ちそうなガイドラインを説明します。

1.3.1 問題ステートメント(Issue Statement)

問題ステートメントとは、これから解決しようとしている問題が何であるかを明らかにしたものです。言い換えれば、モデル開発の目的といえるものです。有用なモデル開発のためには目的を明快にすることはとても重要です。どの様にシステムを改善する必要があるか、どの振る舞いが具体的に問題なのかをはっきりさせずに、システムやプロセスのモデルを開発することは困難です。明確な問題意識を持つことは、モデルを開発し上手く適用して行くには必要不可欠なことです。

1.3.2 変数の特定(Variable Identification)

問題をモデルとしてひと言で表現しようとしたときに、モデルに含まれるキーとなる量を特定します。通常は、これらの多くははっきりと認識することができます。その中でも最も重要なものを識別するためには、認識した変数をすべてリストとして書き出して、そのリスト上でランク付けをしてみると良いでしょう。

1.3.3 参照モード (Reference modes)

参照モードとは、一連の振る舞いとして現れるパターンのことです。参照モード図は、重要な変数の時間的変化を表現するために描かれるグラフですが、必ずしも実際に観察した振る舞いである必要はありません。もっと正確に言えば、参照モードは、関心ある振る舞いの特徴を特定するポンチ絵の様なものです。例えば、ある会社の売上高について言えば、「成長している」とか、「アップダウンを繰り返している」とか言ったものです。「成長トレンドのうえでアップダウンしている」ということもありえます。参照モードには、過去の振る舞いだけでなく将来の振る舞いも含めることができます。それは、「起こるかもしれないと想像すること」かもしれません。「決して起こって欲しくないこと」かもしれませんし、「将来起こって欲しいと切に願っていること」かもしれません。「恐れていること」や「理想的なこと」を時間軸でポンチ絵として描いたものが参照モード図なのです。参照モード図は具体的に定義された時間軸に対して描きますが、問題ステートメントを練り上げて境界をはっきりさせることが目的です。[3]

---

[2] "Guidelines for Model Conceptualization", by Jorgen Randers in Jorgen Randers (ed.), Elements of the System Dynamics Method, MIT Press, Cambridge, MA 1980 pp.117-138。 現在は、Pegasus Communications から入手できます。

[3] 正確性よりはむしろ、時間軸がどの程度のスパン(月単位で直近数年なのか年単位で100年間なのかということ)で描かれるのかが重要です。

1.3.4 実現性の点検

実現性の点検とは、システムに含まれていると考える要素と要素がどの様に関係しているのかを明らかすることでチェックすることです。そのために、ある要素とある要素の間にある関係を文章化します。それを実現性の点検文と言います。実現性の点検文を見れば、システムにはどの様な要素が含まれているのか、あるいはそれらの要素間でどの様に相互作用しているのかといったことについて基本的なことが分かります。ある変数が変化したとき、大きく値が変化する変数がある場合、それがどの様ないきさつによるものかといったことが分かります。実現性の点検は、どの様なつながりがあるのかといったことを単にノートとして記録されたものに過ぎません。また、多くの場合は心に留めおくといったレベルのことでさえあります。実現性の点検文として書き出されるものは、モデル化したシステムについての知識に基づいています。

1.3.5 ダイナミック仮説(Dynamic Hypothesis)

ダイナミック仮説とは、「どの様な構造がその様な参照モードを生み出しているのか?」という仮説です。ダイナミック仮説は、動詞で表現されます。ダイナミック仮説は、因果ループやストックとフローのチャートとして表現します。ダイナミック仮説は、「何がモデルの中に含まれているのか?」また、「何が含まれないのか?」を決めるために使用することもできます。すべての仮説がそうである様に、ダイナミック仮説は常に正しいとは限りません。モデルを作成するためには、精査と修正が必要です。

1.3.6 シミュレーションモデル(Simulation Model)

シミュレーションモデルは、精査された結果として導き出されたダイナミック仮説であり、明示的に表現された数学的関係の集まりです。シミュレーションモデルはシミュレーションによって振る舞いを生み出します。シミュレーションモデルは、構造に含まれる要素の値を様々に変化させて、どの様に振る舞いが変化するかを、コンピュータ・シミュレーションを使って繰り返し確かめることができるので、構造に含まれる要素がどの様に振る舞いを決定するのかを理解するための実験室の様なものと言えます。

上記のプロセスは、試行錯誤の結果として何度も進んだり元に戻ったりするものですから、柔軟に対応しなければなりません。問題を解決する仕事では、それまで正しいと考えていた考え方を変える必要が起こってきます。このモデリングガイドの後半では、上記のプロセスの中で、重点を置くプロセスを順次変えながら、多くの異なる見地からモデル開発のプロセスを経験できる様にしています。

Vensimによって分かりやすく簡単に変数に名前を付け、実現性の点検のための情報を設定し、ダイナミック仮説を設定し、シミュレーションモデルを構築するといった作業を進めることができます。問題ステートメントの作成や参照モードを作成するには、紙と鉛筆や、他の技術でも容易にできるかも知れません。Vensimを使えば、簡単にビジュアルなモデルとしてダイナミック仮説を構築することができます。これは紙と鉛筆でも同じ様に描き出すことができます。しかしながら、シミュレーションするためには、コンピュータが必須となります。

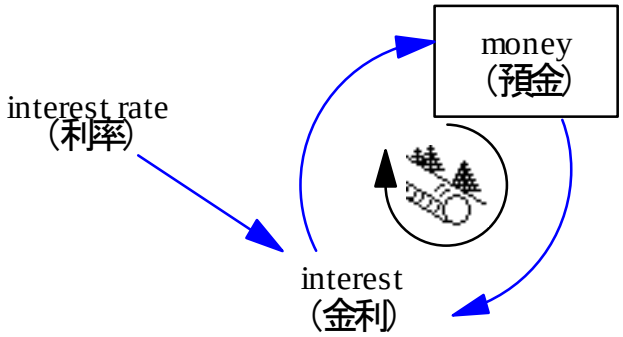
1.4 基本的な構造とその振る舞い(Fundamental Structure and Behaviors)

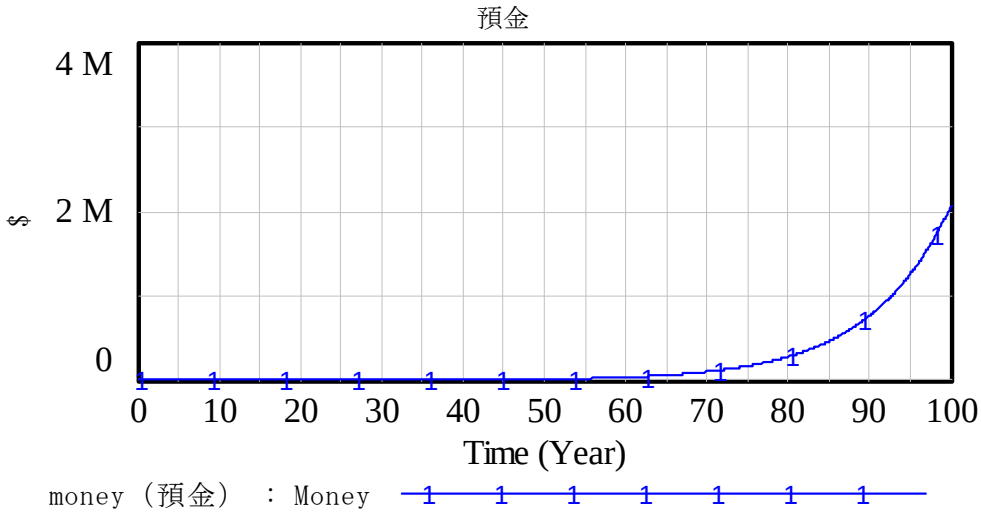
以上議論した様に、モデル構築のプロセスでは、参照モードで示された振る舞いに、どの様な構造が関わっているのかということについて、仮説の設定が求められます。これは大変難しいことかもしれません。しかし、モデル構築の経験を積むに従って次第に慣れてくるものです。なぜなら、構造とその構造が生み出す振る舞いの関係を経験として習得できるからです。この習得の経験の第一歩として、最も単純で多く現れる基本の振る舞いのパターンとそれを生み出す最も単純な構造について学習します。

次のセクションの中で、成長、崩壊、調整と振動の振る舞いとそれらを生み出す単純な構造を見て行きます。それらのダイナミック仮説を生み出す本質的なものはフィードバックです。フィードバックを強調するために、モデルは因果ループ図で示します。

1.4.1 指数関数的成長(money.mdl)

仮に銀行に100ドルを預けたとしましょう。利率が年10%で複利計算として、100年経てば、どうなるでしょうか。これは、１階のポジティブ・フィードバック・ループの例です。



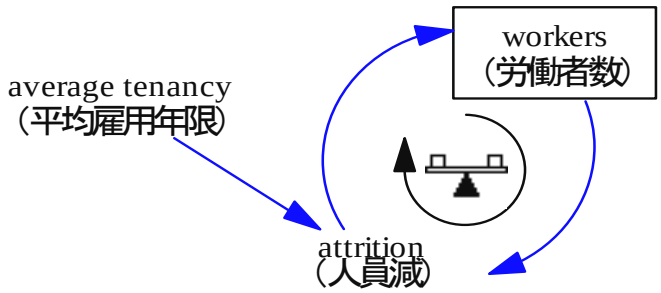


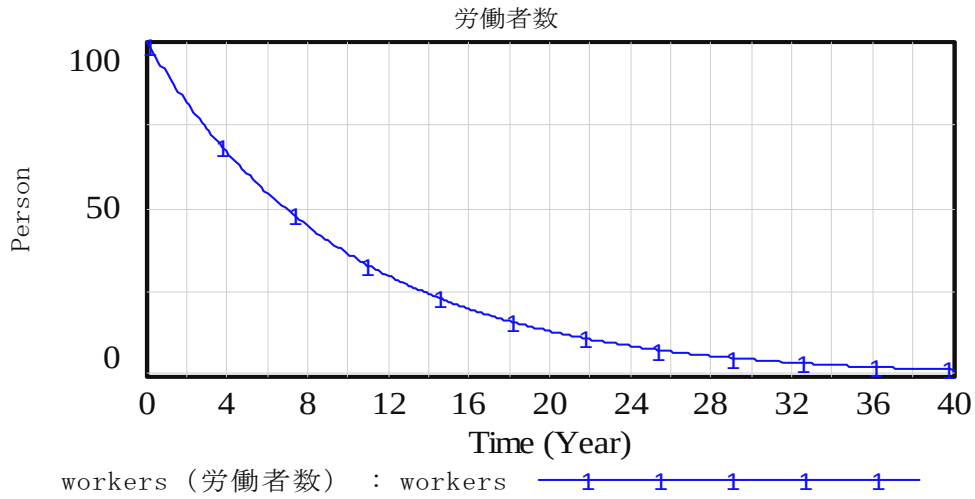
100年経てば200万ドル以上になります。数関数的成長は倍増時間が一定となる性質があります。もし、100ドルが200ドルになるために約7年かかっていますが、100万ドルから200万ドルになるのにもやはり同じだけの約7年かかります。この例を使って、利率と残高が2倍になるためにかかる時間の関係を調べればより理解が深めることができるでしょう。

この例については、誤差を生じない様に、TIME STEPを十分に小さい値(例えば、0.125)を選択するか、積分法としてルンゲ・クッタ法を選択する様にしてください。

1.4.2 指数関数的崩壊(workers.mdl)

いま仮に、あなたの会社で働いている労働者が100人いるとします。今後は新たに雇用しないものとします。そして、労働者の平均雇用期間は10年間とします。労働者数はどの様に変化するでしょうか。このダイナミックスは1階のネガティブ・フィードバック・ループの例です。





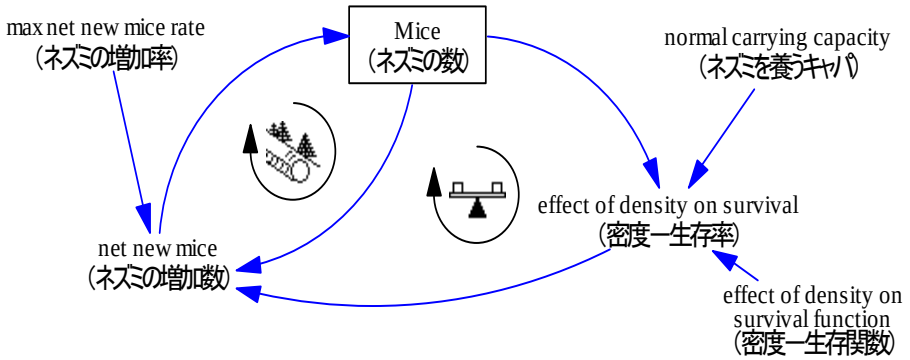
労働者数は、初めは非常に速く、その後はゆっくり減少します。これは指数関数的成長と反対です。すなわち、指数的成長では、初め変化は遅く、その後迅速に変化しました。しかし、指数的崩壊では初めは迅速に、その後ゆっくり減少します。また、指数的成長では指数関数的成長は倍増時間が一定という性質がありましたが、指数関数的崩壊にも同様の普遍的な性質があります。すなわち、労働者の半分が退職して行くのに約7年かかるとす

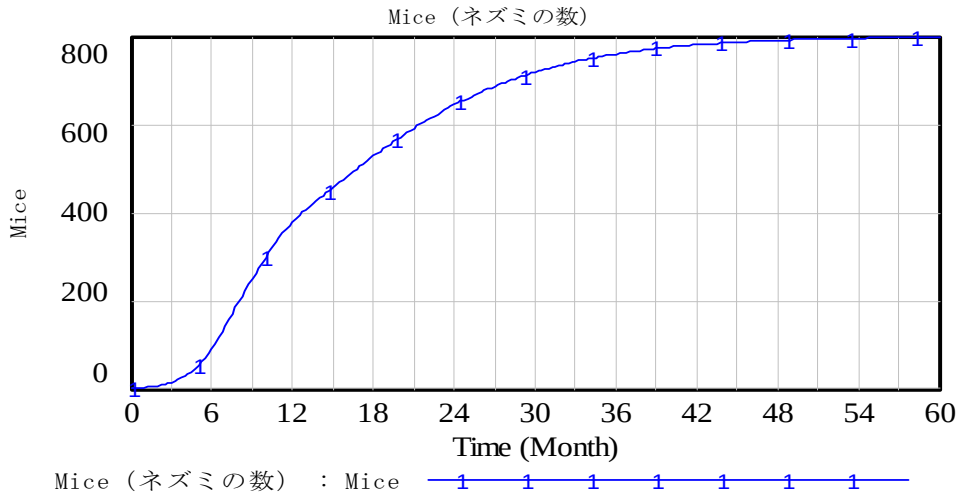
れば、残りの労働者が更に半分になるためには、7年が必要となります。平均雇用期間を色々と変化させて、労働者が半分まで減少するのに要する時間の関係を観察すれば更に理解が深まります。

指数関数的崩壊に関して重要なことは、最終的に値が収斂する値を0から他の値に移しても、同じ様な振る舞いになるということです。仮に室温が10度であったとき、40度に暖房温度を設定すると室温はどの様になるでしょうか。

1.4.3 S字成長(mice.mdl)

いま仮に、家の中にハツカネズミを何匹か放して、殺さずに放置したと仮定してください。ハツカネズミの頭数はどの様になるでしょうか？一見、振る舞いは指数関数的成長の例の様に見えます。確かに、最初の間はそのとおりです。しかしながら、いくら増えたとしても、家の中のハツカネズミが200万匹になることはないでしょう。



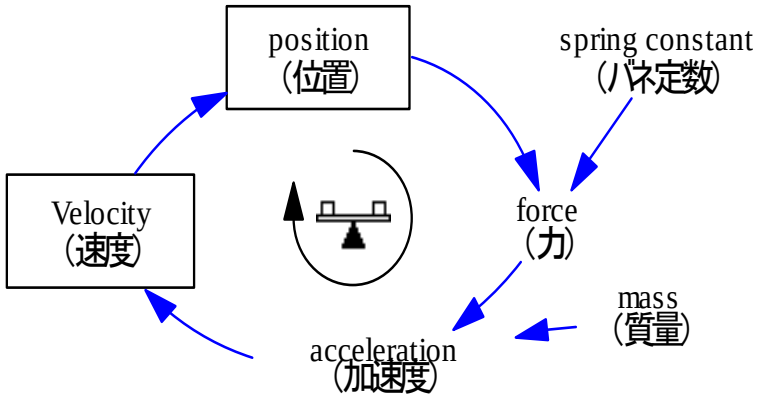


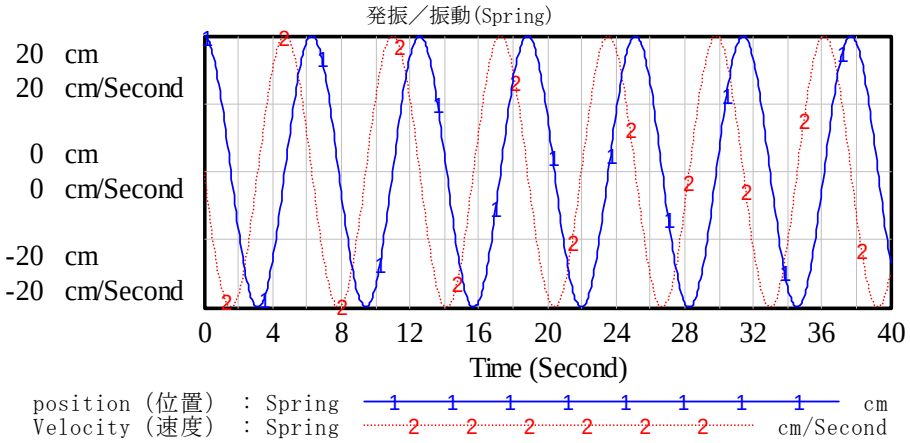
ですから、初めは確かに指数関数的成長の様に見えますが、終盤は指数関数的な調節が働いている様に見えます。このマウス数の推移は単一のフィードバックループでは理解できません。このモデルの場合には、ポジティブとネガティブなフィードバックループが両方存在し、初期にはポジティブ・ループが支配的であり、終盤ではネガティブ・ループが

支配的となっています。この働きを調べるためには定数 max net new mice rateを変化させてみると良いでしょう。値を変化させるにつれて、変数 net new miceの振る舞いはどの様になるか観察してみましょう。このモデル中に含まれる調節のプロセスは、ルックアップ変数 effect of density on survival functionの形に大きく依存します。この関数の形を変化させて実験してみましょう。

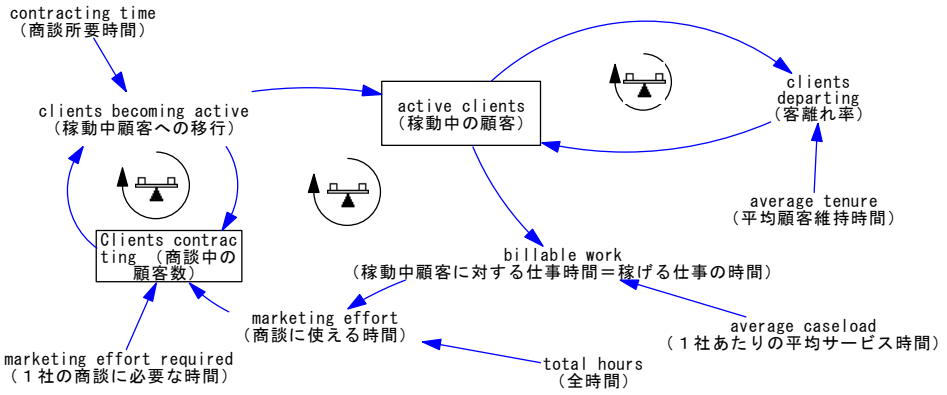
1.4.4 振動(spring.mdl)

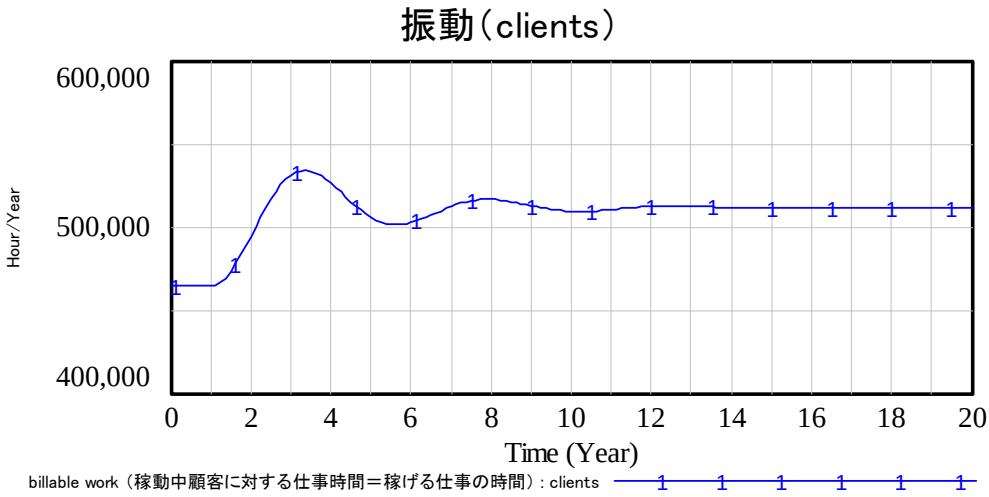
摩擦がないと仮定した水平面上での､おもりがついたスプリングの動きを考えてみよう。





正確な結果を得るために、このモデルをシミュレートする場合はルンゲ・クッタ積分法を必ず使用する様にしてください。この例は、振動を生み出す最も単純な構造です。振動は減衰しません。良く似た問題をもう一つ考えてみよう。コンサルタント業務で、新しいクライアントを獲得するマーケティング業務にかける時間と、アクティブなクライアントから受注する有料業務を行う時間が一定であるという関係に置き換えて考えてみよう。(consult.mdl)





このモデルでは、total hours が時間0においてステップ的に増加したと仮定してシステムを駆動してみます。また、ストックは初期化式を含んでいますが、初期化式に関するリンクを図から隠しています。2つの振動モデルを比較してまず気づくのは、後者の例ではモデル中にループが2つ追加されていることです。実際のところ、ビジネスシステムにおいては、2つ以上のストックが存在するので、単一のループだけで記述できることは大変稀です。メインとなるストックのループに起因するダイナミックスの周辺には、多くの小さなネガティブ・ループが存在することが想定されます。そして、一般的には、これらの小さなネガティブ・ループは、メインとなるループのダイナミックスを減少させる方向に作用します。一度、モデルの定数を変化させて、どの様に振る舞いが変化するかを観察してみてください。

「振動」については、2章の中の労働力と在庫モデルおよび8章において、更に議論したいと思います。

第２章

労働力、在庫と振動
(Workforce, Inventory and Oscillation)

2.1 はじめに

本章では、モデルを開発、分析そして活用する中で課題となる概念についてフォーカスします。本章を学んだ後、VensimでSDモデルを実際に動かしてみるとより理解を深めることができます。

2.2 予備知識

組み立て式のアルミサッシの生産、販売を行っているとします。あなたの会社の事業は全体的に大変順調です。しかし、生産を増加させるシフトをしいた途端に、仕事が少なくなって生産能力を遊ばせてしまうといったことをしばしば経験します。通常は、この様なことが起こったら市場需要の変化や経済の責任にしてしまいますが、いささか疑問は残ります。なぜなら、過去８年間を遡って調べてみると、実際には販売量は生産量よりもはるかに安定しているのです。あなたの課題は、実際には販売量は安定しているにも関わらず、増産シフトをしいた途端に生産能力を余らせてしまう様なことがなぜ起こるのかを明らかにして、その対策を考えることです。

この問題に取り組むにあたって、できるだけ単純化して考えてみましょう。単純化して考えるのには、次に示す様な多くの理由があります。

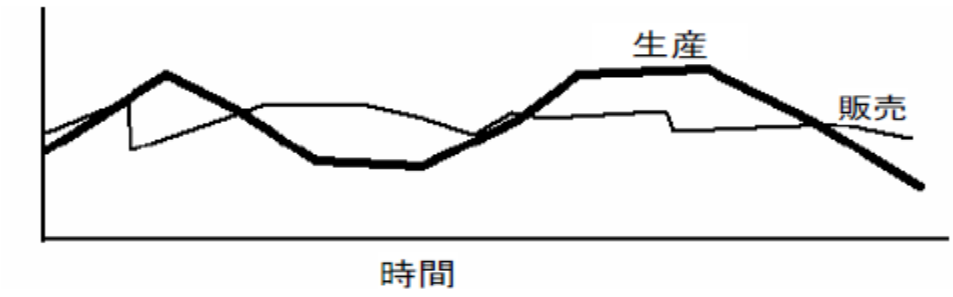
- モデルを理解し易い。
- 結果を速く知って対策の正しさを検証できる。
- 複雑なモデルを完成させて、それから洞察を抽出するよりも、単純モデルで始めて、後から必要に応じて詳細を加える方がより効率的である。
- モデリングの初期には、全体を把握することが大切ですが、モデルを単純化することで、自然に全体を概観できる。

しかしながら、いつも単純なモデルから考え始めることができるとは限りません。多くの場合、問題ある振る舞いというものは複雑さの結果であることが多いからです。そして、あなたは目の前の複雑な構造をそのままモデル化しようとしてしまうのが常です。確かに、Vensimのツールを活用することで複雑さに対処することができるのですが、問題ある振る舞いの背景にある問題の本質を掴み取るまでは、単純性を大切にすることを忘れないでください。

小さな単純なモデルを開発した場合は取り組むべき問題すべてを議論するためには十分に出ないこともありますが、そのプロセスを通して問題に対する理解を深めることが後で役に立ちます。これとは反対に、大規模な複雑なモデルは開発するために非常に長い時間を費やしてしまい、何も得られなかったということは良く起こります。これでは時間の著しい無駄遣いになってしまします。

2.2.1 参照モードで問題の本質を把握する

参照モードは問題をグラフで表したものです。問題を言葉で表せば、「生産は販売ほど安定しない」となるでしょう。これをグラフで描けば次の様になります。



この参照モード図は、「これから作成するモデルが示すと考えられる振る舞いをスケッチ

したもの」ということもできます。それは、記録に基づく現実のデータかもしれませんし、新しい状況ではこういうことが起こるかもしれないというあなたの想像や期待かもしれません。参照モードは、問題に集中することによって、無駄なことを考えたり試行錯誤したりしない様にするために作成します。参照モードを作成したならば、この参照モードに質的に似た振る舞いのパターンを生み出すと思われる、最も単純な構造を定義することが重要です。作りあげた構造が適切であったなら、モデルを改善することによって、入手したデータによって定量的にモデルの有効性を検証することができるはずです。

2.2.2 実現性の点検で問題を具体化する

ビジネスがどの様に進められるかということに関する常識をモデルの中に入れ込みます。例えば次の様なものです。

・労働者がいなければ、生産はできません。
・在庫がなければ、出荷はできません。
・販売がある期間増加すれば、生産を拡大しようとします。
・在庫を持たなければ、売上を増やすことはできません。

ダイナミックス仮説に基づいてシミュレーションモデルを構築するときは、これらの実現性の点検に関する情報を、常に心に留めておくことが重要です。

2.2.3 ダイナミック仮説で構造を考える

ダイナミック仮説を置くということは、どの様な構造ならば参照モードに示された振る舞いを生み出し得るかということを考えることです。この例では、単に参照モードで示されている2つの変数（生産と販売）がどの様に関連しているかを考えることにより、ダイナミックス仮説を定式化することができます。つまり、変数「販売」が与えられたとき、変数「生産」を決定するためのポリシー(あるいはルール)を特定することです。この会社のダイナミックス仮説は、マネージャーは現在の販売高に基づいて「生産」量を決める際に、必要以上に高い(あるいは低い)生産量を決めてしまっているのではないかということです。

参照モードには生産と販売という2つの変数が示されていますが、これら2つの変数をモデルに含めるのが自然でしょう。一般的に参照モードを作成することから始めるのが良いと言われているのですが、今回のケースにおいても生産、販売という変数を中心に考えることになったことは、このことの1つの証左といえるでしょう。

2.3 労働力/在庫モデル(wfinv1.mdl)

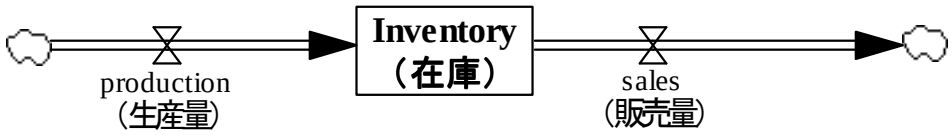
変数「生産」と「販売」にはどの様な関係があるのでしょうか。何かを売る前には、商品を生産する必要であるという意味で密接な関係があります。販売と生産は2つの方法で関連づけられます。すなわち、

①物理的方法:　生産は販売するための商品を生み出すこと。

②情報的方法:　管理者は売上に基づいて生産に関する判断を下す。

まずは、物理的方法に基づいてモデル作りを始めましょう。生産された時点では、商品はすぐには販売されません。すなわち、販売されるまで在庫として保管されます。そして、販売された時点で、商品は在庫から引き落とされることになります。一般的には、在庫となるケースだけではありません。需要に対して供給不足の場合は、生産が行われると在庫となることなしに直ぐに出荷されバックログが減少することになります。なので、販売と生産が分離するためには、在庫あるいは在庫とバックログの組み合わせが必要になります。バックログがモデルの中で使用される場合は、販売の代わりに、注文および出荷という概念を考慮するのが良いでしょう。

このモデルでは1つのストック変数Inventoryを使用することにします。そして、そのストックに対するインフローとアウトフローを加えます。次に、マージツールを使用して2つの変数をドラッグして、バルブのうえにproductionとsalesをマージします。

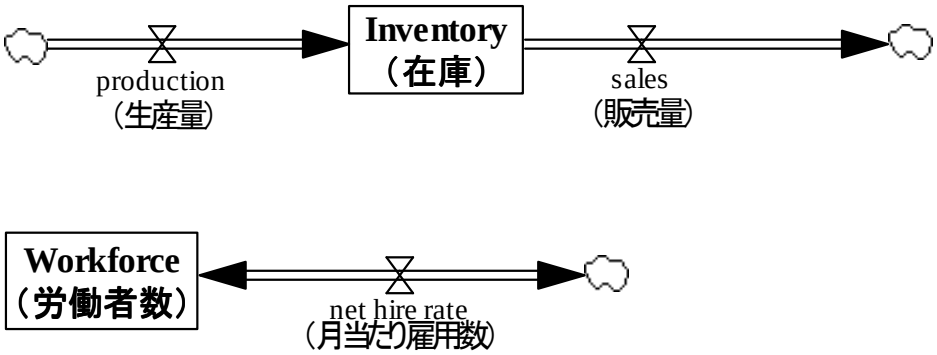


ここでは、Inventoryをモデルに加えることに決めた「後に」、フローとして既存の変数を付け加えるという段取りが重要です。すなわち、初めにInventoryというストックを導入するということを決めました。そこで、ストックを表す長方形図形を作成すると同時にInventoryという名前を付けました。その後、そのストックに出入りする2つのフローを図として書き加えました。そして、書き加えたフロー対して、既に定義してあった変数を使って、フローに名前を付けることができたという、一連の段取りに注目する価値があります。何か問題を処理するために、まずモデルの下絵を描いてからコンピュータ入力して清書するという段取りでは、ひらめきを活かしながら問題を処理することはおぼつかないからです。

2.3.1 労働力

次に、productionがどの様にして決定されるか考える必要があります。長期的な投資を考えるときは生産能力が大変重要であり、産能力は大変安定しているものです。より短期的には、より多くの人々が雇われたり生産シフトが図られたりします。そこには、こまごまとした考慮事項が付け加わってきます。すなわち、変更がいつ加わるのか、管理も変わるかもしれませんし、メンテナンスの予定も変わるかもしれません。しかしながら、とりあえず近似的に、より多くの人々がいればより多くの製品を作ることができると言うことができます。スタートとしてここから始めてみるのは良いことです。ストックWorkforceをモデルに加えます。Workforceを変化させるものは、労働者の雇用、一時帰休、解雇そして定年退職があります。ここで、単純性が重要であるということを思い出して、これらを

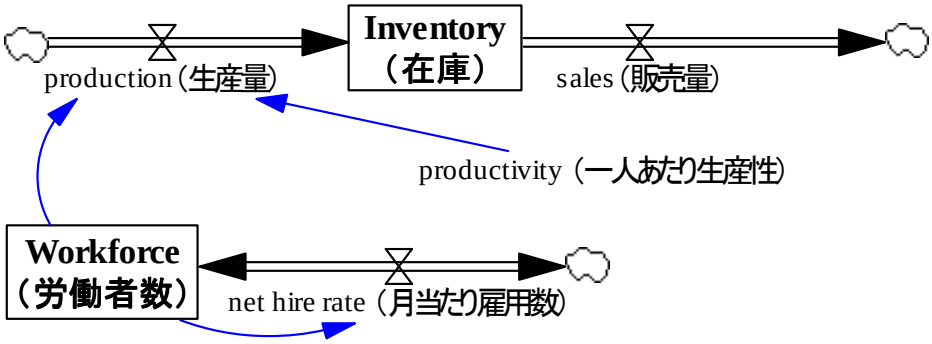
すべて組み合わせて、一つの合成概念net hire rateと置きます。net hire rateは、Worikforceを増加させるだけでなく減少させる場合があることに注意してください。



2.3.2 振る舞いに関する関係（情報）

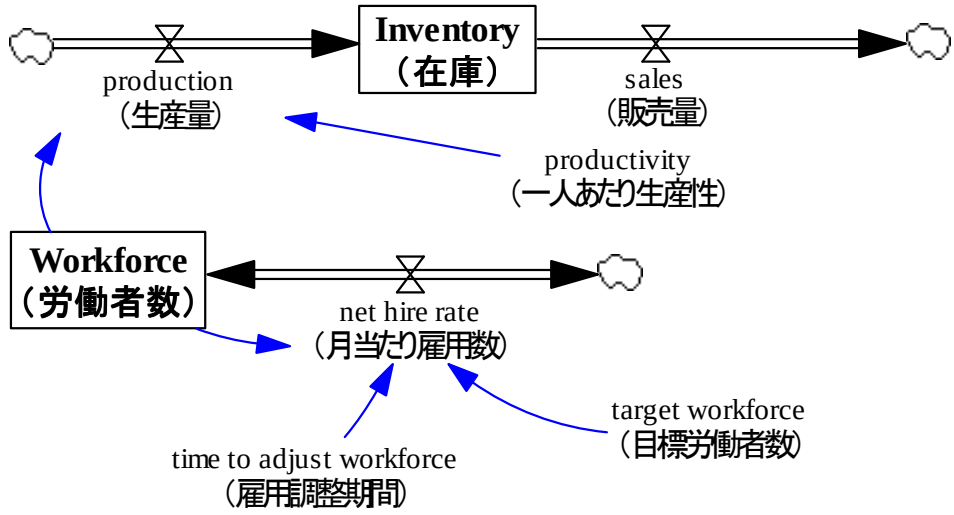
問題の物理的側面を考えることで上記の図を導くことができました。次に情報の連結を定義して行きましょう。ある要素が別の要素の振る舞いに影響を与えるという意味で振る舞いに関する関係づけを考えて行けば良いでしょう。良いモデルを作るためには、重要な物理的ストックとフローを描くことから始めるのが良いでしょう。このようにしてシステムの中心的な部分を固め、それに基づいてシステムの他の部分を順次概念化してゆくことにより、単純化できるのです。因果ループ図を描いたうえでストックとフローに書き換えても良いですし、方程式に直接書き下しても構いません。更には、ストックとフローを描くことから始めて、もう一度、因果ループ図を描たくなるということもあります。何が一番良いアプローチであるかは、問題が何であるかということと、誰が問題分析を行うのかということによります。従って、このガイドでは、章毎に意図的に異なるアプローチをとる様にします。

情報の連結を完成させるにあたって、できるだけ物事を単純化する必要があります。生産について考えるにあたって、生産シフトや装置に関する話はしまい込んで、productionはWorkforceに比例するということだけで表現する様にします。そのために、比例定数productivityを付け加えます。加えて net hire rateはWorkforceの値に依存して決まるものとします。これらを図示すると次の様になります。

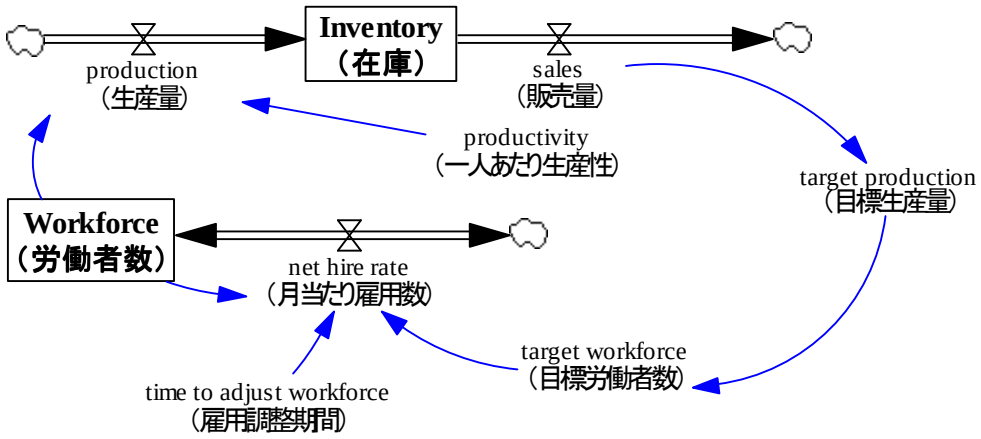


net hire rateは、雇用される正味の人数です。このことを定式化する最もストレートな方法がストック調整プロセス(A stock adjustment process)です。ストック調整プロセスとは、変数(通常はストック)の現状の値と、その目標値あるいはこのようにあって欲しと希望するレベルと比較し、2つの差に基づいた処置をします。例えば、時速40Kmで自動車を運転したいとします。時速50Kmにしたければアクセルを踏みます。あなたの自動車の速度(速度はストック)は、あなたがアクセルをどれくらい踏むかということと、運転している車によるでしょう。

このストック調整プロセスをモデル化するために、変数target workforceとtime to adjust workforceを加えます。そして、それらを結合すると次の様になります。



time to adjust workforceは、マネージャーが労働者の人員数を変更することを決定し求職者を選考したりレイオフを通告したりする時間に相当します。target workforceは、生産に必要となる人員数であり、ストックWorkforceの初期値としてこの値が初めにセットされます。次に、target productionという概念を付け加えます。そして、それをtraget workforceと結びます。Target productionはsalesを基に決められます。



これが、完成した概念モデルです。しかしながら、これにはまだ省略した部分がありシミュレーションできません。次のステップは、概念モデルからシミュレーションモデルを

開発することです。

2.3.3 方程式 wfinv1.vmf

このモデルの方程式は次の様になります。

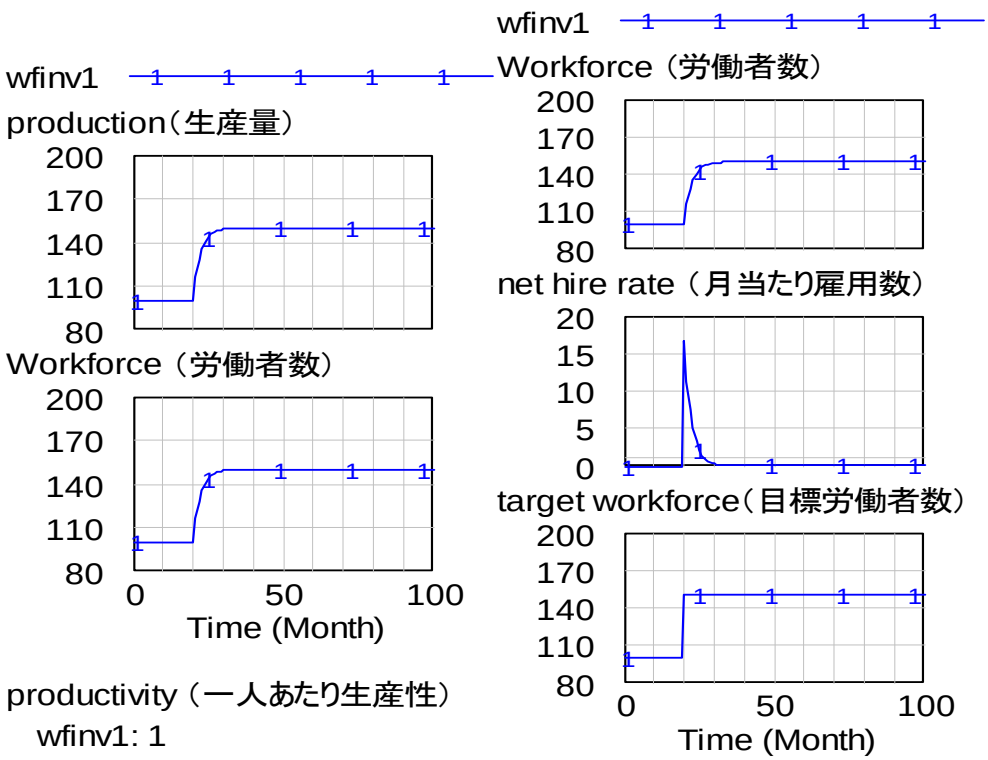
```
FINAL TIME  = 100
Units: Month
INITIAL TIME  = 0
Units: Month
TIME STEP  = 0.25
Units: Month
SAVEPER  = TIME STEP
Units: Month
Inventory = INTEG( production-sales, 300)
Units: Frame
net hire rate = (target workforce-Workforce)/time to adjust workforce
Units: Person/Month
production = Workforce*productivity
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
```

それぞれの方程式は、最初に行った概念化の議論と合う様にしています。salesを定義する式は、Time=20までは100で一定であり、ステップ状にアップし、その後は150が保たれます。この入力パターンはシステムが本当に均衡状態にあるか否かをテストしたうえで、新しい条件(Salesが150に増加したこと)に対して調整して行く様子をチェックするために用いられます。この様なステップ入力は、この例で確認しようとしている様な内部的に生み出されるシステムのダイナミックスを観察するための最もシンプルなパターンです。そして、他に外乱がないときに単純な衝撃がどの様にシステムの中で波紋を広げてゆくのかを観察できるのです。

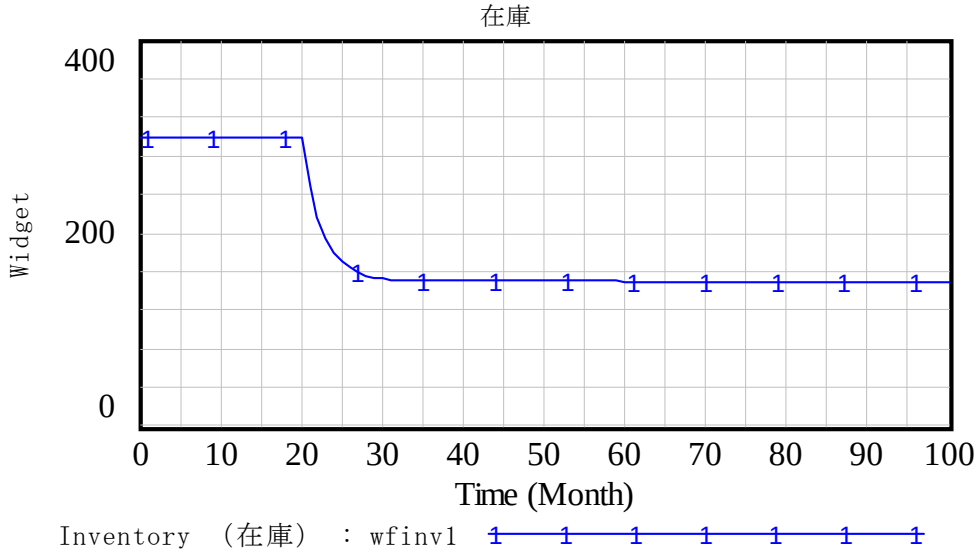
モデルはこれで完成しました。いよいよシミュレーションに名前wfinv1を付して実行します。

2.3.4 分析

変数productionとWorkforceに対する直接原因グラフは、次の様なダイナミックスを示します。



productionやWorkforceはあまり変化しないことは直ぐに分かります。100単位から150単位まで非常に滑らかに調節しています。

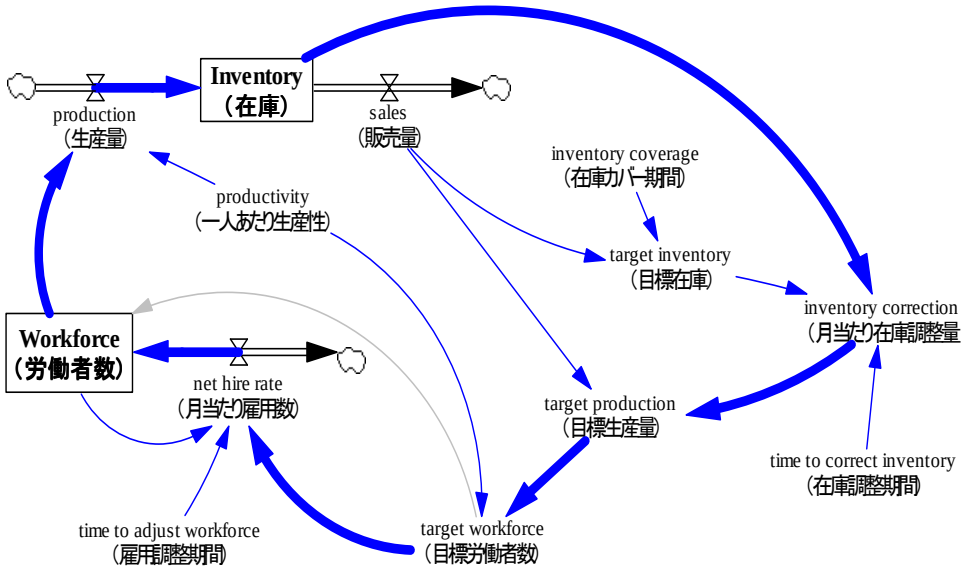


在庫は、初期値300から150まで滑らかに減少します。在庫を持つ目的は、お客様が直ぐ

に製品を適切な構成で利用できる様にするということにあるとすれば、この様な振る舞いは何か間違っています。製品の品揃えを満足しかつ安心できる在庫水準から乖離していれば修正する必要があります。しかし、このモデルでは、その様な修正が行われていません。

2.4 モデルの改善(wfinv2.vmf)

モデルを改善するために、target inventory、inventory correctionと2つの定数を導入します。その考え方はシンプルです。target inventoryは販売からの期待に基づいて決められる目標在庫です。inventory corectionは目標在庫を修正するものです。新しいループを書き足したうえでその矢印を強調しておきます。



2.4.1 追加した式

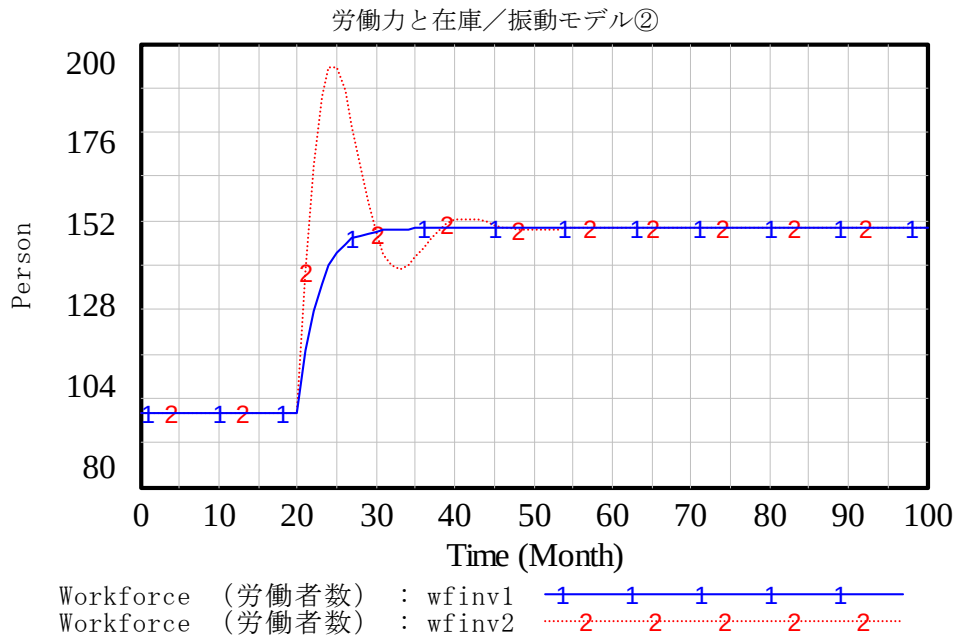
target production = sales + inventory correction
Units: Frame/Month
inventory correction = (target inventory - Inventory)/TIME TO CORRECT INVENTORY
Units: Frame/Month
TIME TO CORRECT INVENTORY = 2
Units: Month
target inventory = sales * INVENTORY COVERAGE
Units: Frame
INVENTORY COVERAGE = 3
Units: Month

net hire rateと同様に、inventory correctionはストック調整プロセスを用いて定式化しています。time to correct inventoryは、著しい在庫変動や生産におけるスケジュール変更の必要性に気づくために必要な時間に相当します。

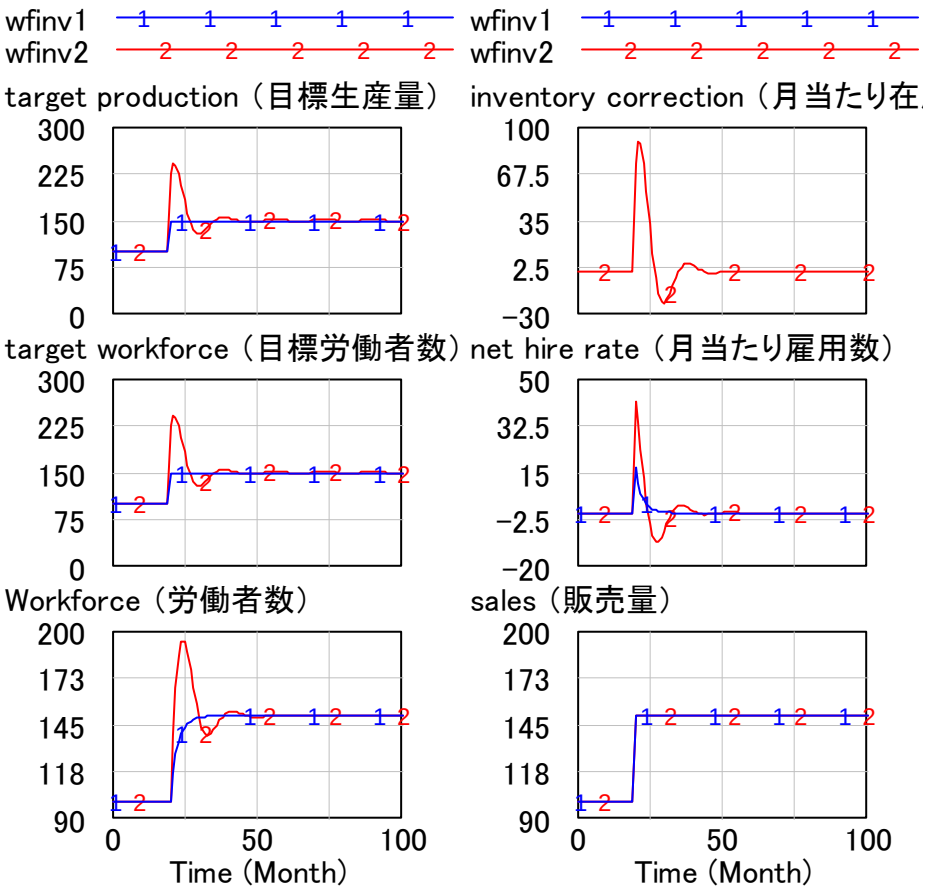
ここで行ったinventory correctionの定式化とnet hire rateの重要な違いは、直接ストックに影響を与えているか否かということです。net hire rateはWorkforceを調整しようとします。そして、inventory correctionはtarget production、target workforce, net hire rate、Workforce、productionを経て最終的にはInventoryにも影響を与えます。この関係の連鎖にはストックWorkforceが介在し、そのことが重要なダイナミックス上の結果をもたらします。

2.4.2 改善後のモデルの振る舞い

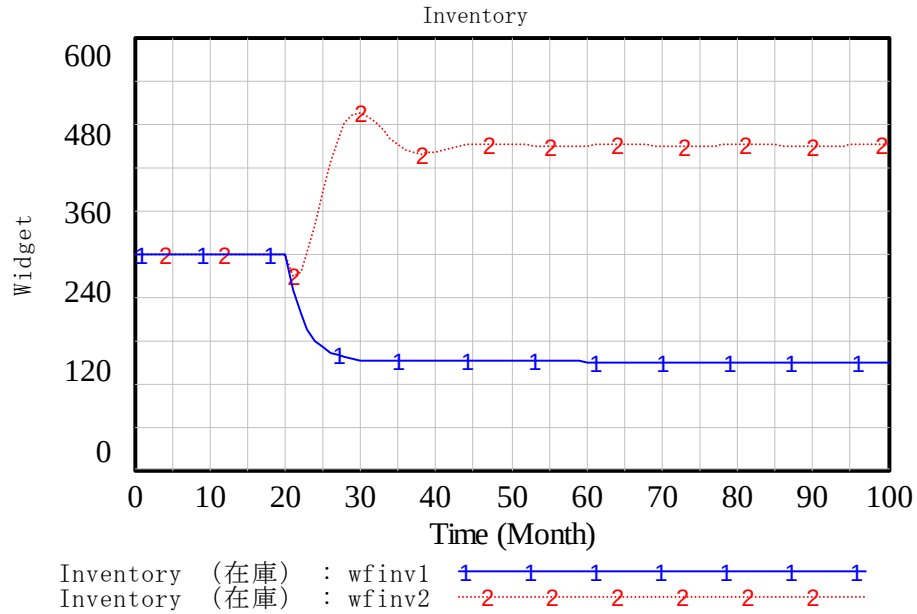
モデルを実行名vfinv2でシミュレーションします。 最初にWorkforceの振る舞いのグラフを作成します。データセットは、wfinv1およびwfinv2の両方を読み込みます。



その振る舞いは先のモデルとは全く異なっています。新たなモデルのWorkforceはあまり安定化せず振動が発生しています。Workforceに対して直接原因グラフを用いてトレースして見ましょう。また、target workforceとtareget productionについてもトレースを行ってみましょう。結果は次の様になります。

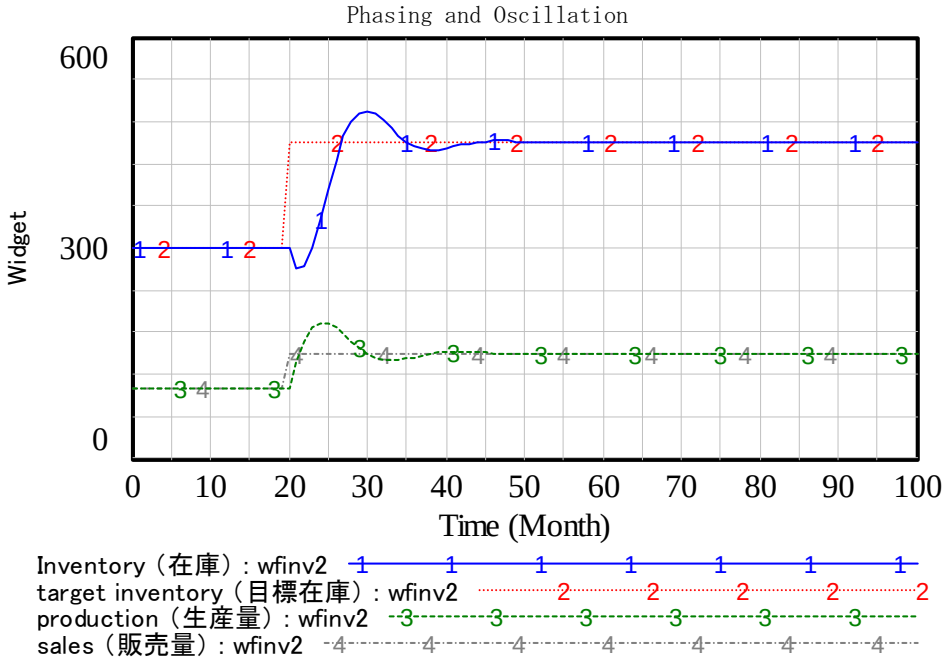


inventory correctionは、wfinv1のモデルにはありません。それゆえ、wfinv1はプロットされていません。target porctionの振動する振る舞いは、在庫水準修正のためです。「在庫」のグラフも同じ様な振動を示しますが、wfinv1とは違った値を示します。販売が増加するとき、在庫水準はwfinv1の様に下がるのではなく、新しいより高い値を望ましいレベルとしてそちらの方に向けて調整が働きます。



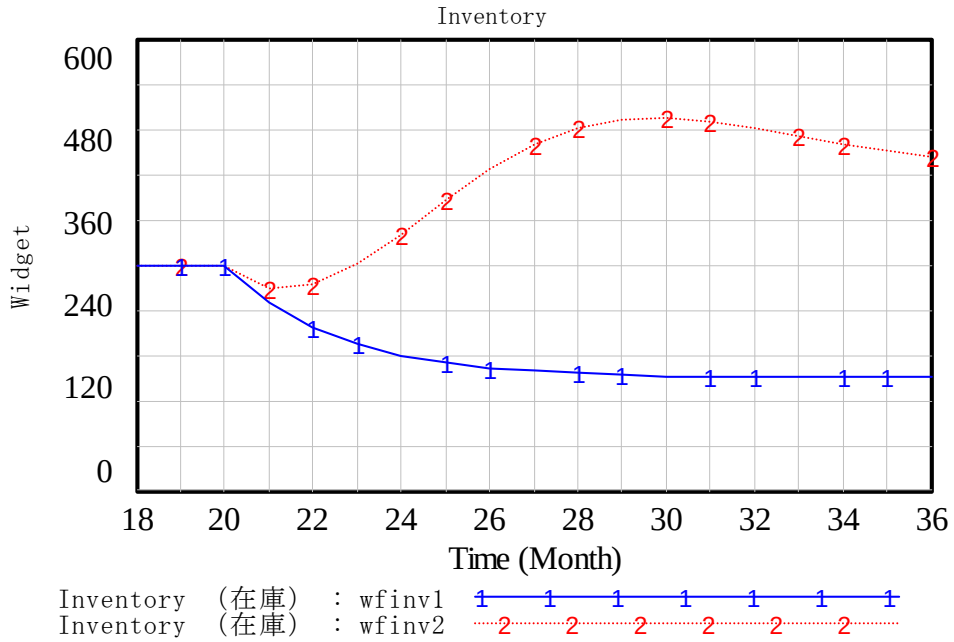
2.4.3 変化のフェーズと振動

このモデルを更に深く考察するために、コントロールパネルのグラフタブを用いて、Inventory、target inventory、production、salesのグラフを作成してみましょう。時間スケールを最小0から最大600にセットします。そして、コントロールパネルのデータセットタブを使って、データセットwfinv2が先頭になる様にしてください。



販売量がステップ状に増加した後の変化に焦点を当てたいので、シフト・キーを押しな

がらグラフの興味のある部分をドラッグしてみましょう。または、コントロールパネルの時間軸タブから、18ヶ月目から36ヶ月目にフォーカスしてみましょう。次の様なグラフが得られるはずです。



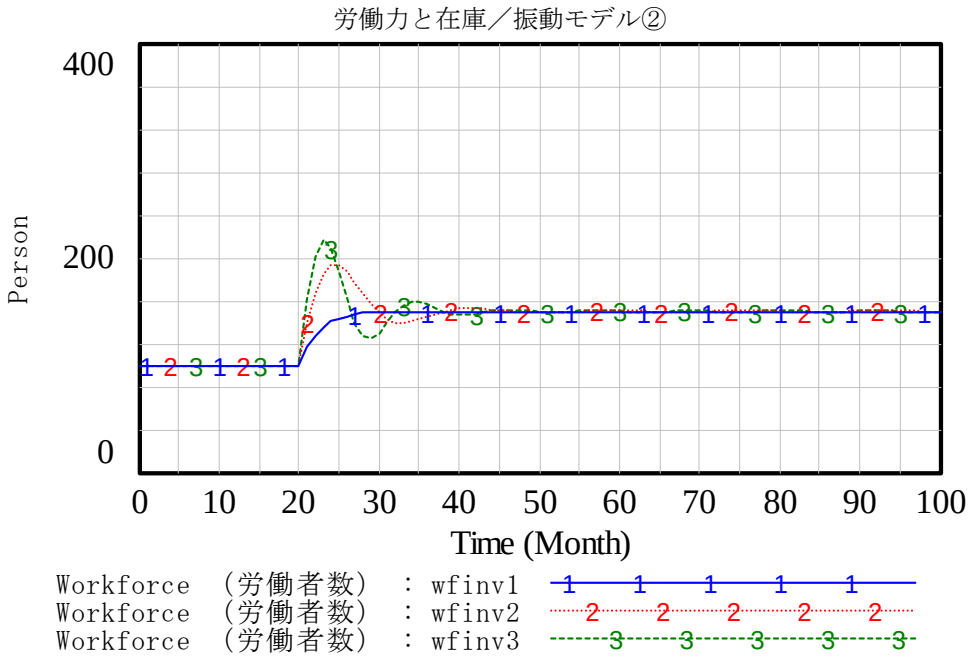
salesがステップ的に増加すると、salesがproductionを上回っているため、Inventoryは下がり始めます。salesが増加するに伴って”target productionが増加すると、その後、productionが徐所に増加します。何故ならば、Inventoryが減少すると、inventory correctionが増加するからです。しかしながら、新規労働者を雇用するために必要な時間のためにproductionを増加させるプロセスに遅れが発生します。結果的に、21.3ヶ月目に、productionはsalesと等しくなります。その結果、在庫の低下は止まり、徐所に増加し始めます。

Inventoryが増加しproductionも増加します。Inventoryがtarget inventoryを下回っている場合はproductionを増加させる圧力があります。しかしながら、productionがsalesを上回っていてもproductionが増加し続けます。net hire rateがマイナスになる前に、現状ある差がinventory correctionからの圧力をバランスさせるためにも、実際のところ、productionは、充分にsalesよりも多い必要があります。Inventoryがtarget inventoryに等しいとき、productionはsalesよりも多い状態にあります。このことに注意する必要があります。なぜならば、労働力は余っており、引き続いて在庫の増加の原因になっているのです。salesよりもproductionの変化が大きくなっています。即ち、モデルは、先に参照モードで描いた様にsalesよりもproductionの変化が増幅することを示しています。

実際のところ、販売の変化が生産に対して伝わるとき、変化が大きくなることは物理的に避けられないことです。一時的に生産の増加量は、販売の増加量を上回っている必要があるのです。なぜならば、生産量の調整がなされる前に、(販売の増加によって) 減少してしまった在庫を回復する必要があるからです。そして在庫を補充したうえで、より高い目標レベルに在庫を積み上げる必要があるのです

2.4.4 変化への対応の感度

パラメータを変更させてモデルで実験することは興味深いものです。会社のパフォーマンスを改善するための施策の一つは、在庫の変化をより積極的に修正することでしょう。ここで、time to correct inventoryをより小さな値にしてシミュレートしてみましょう。いま、time to correct inventoryを2ヶ月から1ヶ月に減少させる(wfinv3)とシミュレーションの出力productionは次の様になります。



ここで、在庫を修正する際の遅れを小さくすることで、確かに期間は短くすることができるのですが在庫量の振動の振幅を増加させてしまうのです。実際の会社でこの様なことが起こることはあまり良いことではありません。

このモデル化の演習では全く直観的にこのレスポンスを説明します。システムは在庫を積み増すためにオーバーシュートする必要があるのです。在庫をより速く積み増すためにはよりオーバーシュートの量を大きくする必要があります。システム内部に含まれている遅れのために、オーバーシュートは在庫の積みすぎを起こしてしまいます。その次に、そのことは生産を下げる様に働きます。オーバーシュートが大きくなれば、その減少もまた大きくなります。

2.5 発展問題

今まで作成してきたモデル構造では、実現性の点検「在庫がないと出荷できない」を満足していないことが分かります。新たに手を加えて、この欠点を修正するためのモデル構造を開発してみてください。その際に、明示的に出荷を表す変数と出荷の制約を表す非線形関数（通常、表関数が使われます）を使ってみると良いでしょう。

第３章
プロジェクトダイナミックス

3.1 はじめに

人間の活動には、大学の期末レポートを書くことから原子力発電所の建築に至るまで、非常に良く似たダイナミックスを見出すことができます。プロジェクトには、最初に設定した目標、目標達成に向けた多くの仕事、そして目標の無事達成があります。　プロジェクトにおいては、目標の未達成、予算超過、納期遅れ、低品質といったことが起こり得ます。

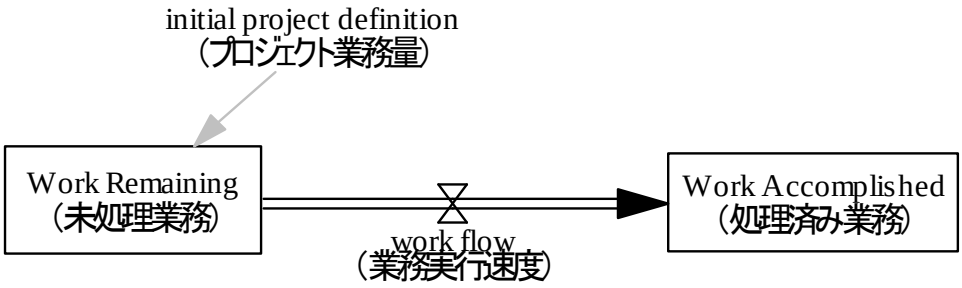
本章では、プロジェクト実行のプロセスを理解するためのモデルを開発します。モデルは新製品を設計する、プレゼンテーションを準備する、ソフトウェアを開発する、スペースシャトルを建造するといった活動にもそのまま適用できる汎用性の高いものですが、ここでは、分かりやすさのために新しいビルの設計を例にとって話を進めます。

モデルを概念化し作成する際に、イテレーティブ(反復)・アプローチを用います。最も単純な構造から始めて、モデルを段階的に改善して行きます。そうすることによって、大規模で複雑なモデルが出来上がったにもかかわらず、シミュレーション結果は意味をなさないといった状況に陥ってしまうことを防ぐことができるので、イテレーティブ技術は有用です。イテレーティブ・アプローチでは、モデルに変更を加えるたびにシミュレーションを行い、新しく付け加えた部分の効果を観察して確認する様にしてください。

モデル開発では、モデル構造を変更し、そのシミュレーション結果に基づいて、その次の進め方を決めて行くので、いきおいコンピュータを使った開発になります。コンピュータはこの様な作業には向いているのですが、コンピュータを使いこなして作業を進めるために、私たち自身が作業の目的や課題は何であるのかを常に意識しておくことはとても重要です。シミュレーションを実行するときは、どの様な結果が得られはずであるかということをあらかじめ自問自答してからシミュレーションを実行すべきです。もし意外な結果の場合は、なぜその様になったのかを必ず考える様にしてください。正しいと思われる結果が出た場合も、それが正しい理由によってもたらされていることを確かめる様にしてください。あなたの恣意が入らない様に、シミュレーションの実行結果に関する予測を前もって書き留めておくことをお薦めします。経験を積んだモデル開発者の多くは、実行時に発見した意外な振る舞いとそれに対する洞察をノートに書き留めているものです。

3.2 タスクの完了 (project1.mdl)

あらゆるプロジェクトの一番基本的な特徴は、「何かすることがあり、それは少なくとも部分的には実行」されるということです。これをモデル化して2つのストックとその間のフローから始めます。

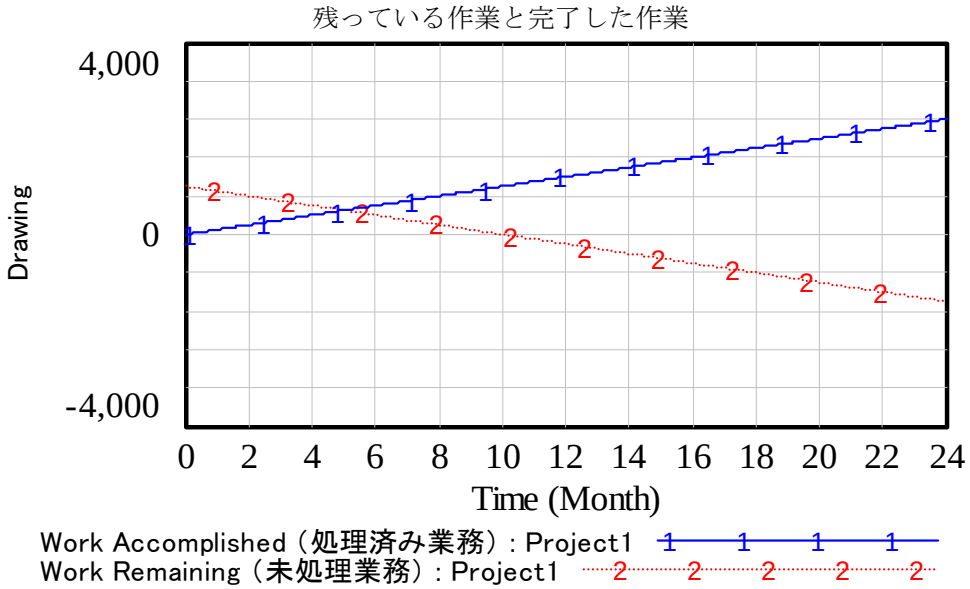


これだけで既にモデルとして完成しています。initial project definitionを1000とおきます。そうすれば、これは10か月かかるプロジェクトということになります。完了時間を確かめるためだけに、24か月間プロジェクトを実行してみます。その際にTIME STEPを0.0625とします。これは1か月の16分の1の「刻み」で計算するという意味です。これは、プロジェクト活動の変化を捉えるには十分な精度です。方程式は次のとおりです。

```
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
```

単位 (Drawing：設計図面の意味) はビルの設計のケースでは適切です。その他のケースでは、タスク、ソフトウェアコードの行数等、他の単位を自由考えて使うことができます。ここでは、プロジェクトというものを全体として取り扱っているということを忘れてはなりません。実際には、プロジェクトは同じ大きさの仕事の集合ではなく、大きいもの、小さいものが組み合わされています。式では平均を表しているとします。実際の問題に適用するときでも、このことを心に留めておくことは(物事の本質をシンプルに表現するという意味で) とても重要なことです。

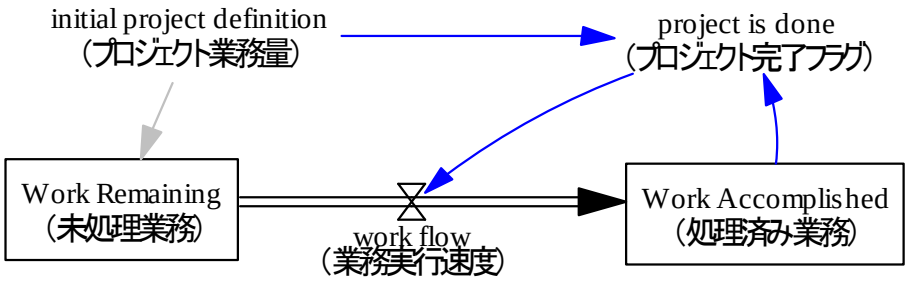
このモデルのシミュレーションを実行すると、次の様なグラフになります。



10か月後に、Work RemainingはOになります。しかし、このシミュレーションではWork Remaining減り続けます。プロジェクトを止めるためのロジックがないからです。

3.3 作業の停止(project2.mdl)

合理的にプロジェクト・モデルを止めるための方法は2つあります。1つはプロジェクトが終わる時点で単にシミュレーションを止めることです。これはFINAL TIMEを決定する式を指定することで実行できます。ここでとったアプローチは、もう1つのアプローチであり、プロジェクトが完成した時点で活動を止めるという考え方に基づいています。そのために、project is doneという概念を付け加えます。そして、project is doneからワークフローまでを矢印で接続します。



project is doneが、Work Remainingではなく Work Accomplishedに依存させることは、スケジュールや予算上の問題がある場合に、プロジェクトの範囲を小さくする判断するためのメカニズムとして役に立ちます。そのために、次の式を追加します。

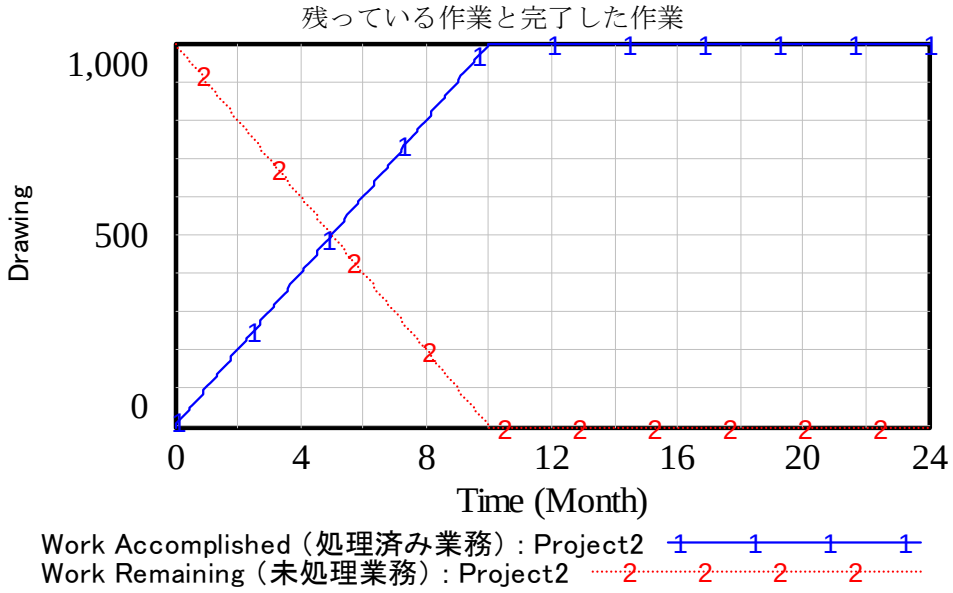
project is done = IF THEN ELSE(Work Accomplished >=initial project definition, 1, 0)

Units: Dmnl

work flow = IF THEN ELSE(project is done, 0, 100)

Units: Drawing/Month

このシミュレーション結果は次の様になります。



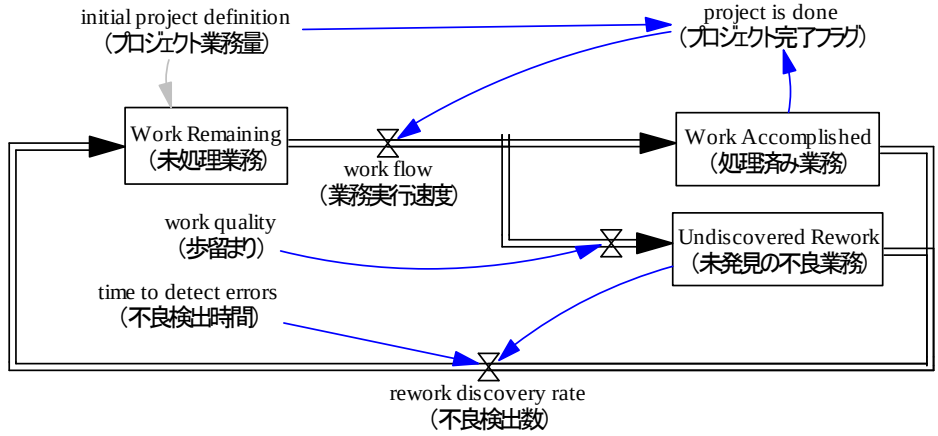
スケジュールどおり終了したという結果です。この様なことはプロジェクト・マネージャーの夢です。　この基本メカニズムは、ほとんどのプロジェクト管理ソフトで使用されています。

3.3.1 積分法

次に進む前に、積分法に関する注意点を説明しておきます。このモデルには、急に作業が停止する部分があります。　この様なプロジェクト・モデルは、オイラー積分法が最適なのですが、通常は、ある種のオーバーシュートが起こります。このモデルでも、後ほどそれを観察することができます。例えば、作業の量が101%まで行ってしまうかもしれません。このオーバーシュートは問題となることはなく、概念としてもそれほど重要ではありませんが、見た目の理由で修正したいこと思うことが時々あります。本章では、細かい計算の話ではなく、重要な概念上の問題に注目する様にします。積分法についてのより詳細については、8章を参照してください。

3.4 エラーとリワーク(project3.mdl)

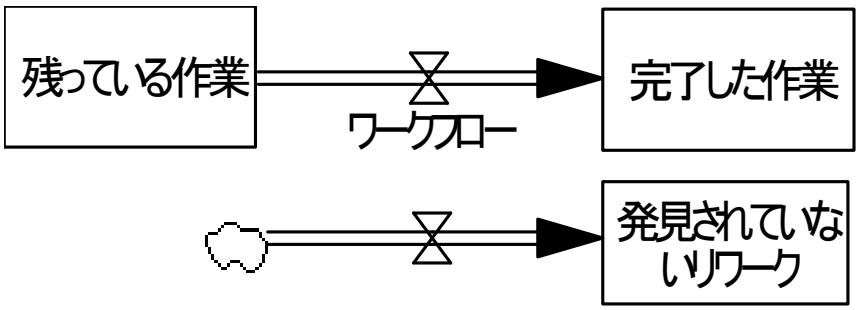
これまでのところ作業はエラーなしで実施されると仮定していました。一般に、この仮定は成り立ちません。エラーは、人的な問題、技術的な見落とし、ポカミスやコミュニケーション不足等、色々な原因で起こります。そして、エラーが生じても、それらを直ちに発見することはできません。エラーを見つけ出すための調査や、モジュールを統合する作業を実施するまで、エラーは発見されないままになってしまいます。



Undiscoved Reworkに蓄えられて行くエラーを含んだ並行の作業のフローと、Work AccomlishedからWorkRemaingに戻るフローを示すダイヤグラムを書き加えました。(もう1つの表し方は、上手く行った作業と上手く行かなかった作業とにワークフローを分割することでしょう。そして、上手く行かなかった作業のワークフローをなされるべき作業へ戻してやることです。もし、これをやったとしたら、完了した作業という概念は、上手く行った仕事と上手く行かなかった仕事の合計と置き換える必要があります。)

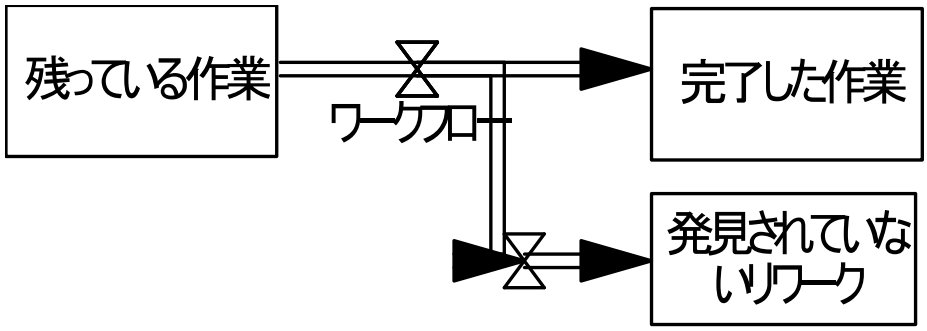
3.4.1 ダイヤグラムの作図

上記のダイヤグラムを描くためには次に示す手順に沿ってください。まずは、ストックとフローだけを書きます。



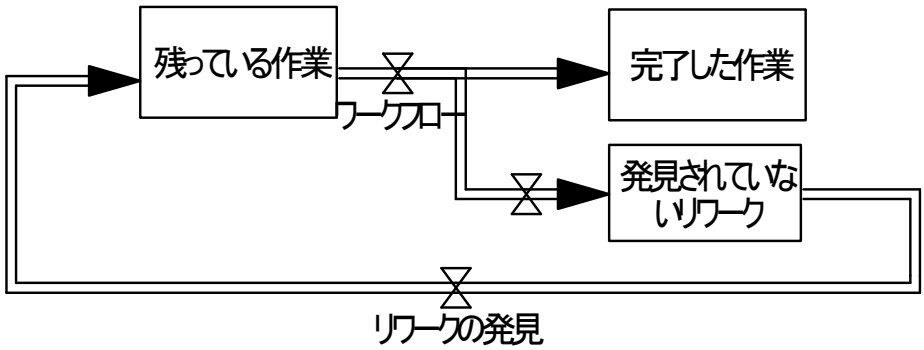
名前のついていないフローを書くためには名前を入力する代わりにエスケープ・キーを押下します。「雲」図形を取り除くために、消去ツールを使います。名前のついていないバルブを、上に示した様に最初に置かれた場所の右に動かす必要があります。フロー・ツールを選択して、「ワークフロー」という名前のついたバルブをクリックしましょう。右に動

かし(下ではなく)、シフト・キーを押下したままにします。そうして、もう一度クリックして、ちょうど名前のついていないバルブのときの様に、こんどは真っ直ぐと垂直な位置に動かし、シフト・キーを押したままにしまにして、そうしてもう一度クリックします。最後にシフト・キーを放して、名前のついていないバルブをクリックします。そうすると、次の様になるはずです。



図は、2つのバルブの相対的な配置がもう少しましになっているかも知れません。不要な矢印を削除するために、矢印ツールを選択して、右クリックしてください、あるいは矢印の上にカーソルを置いてコントロール・ボタンを押下しながらクリックしてください。そして、ダイアログのアローヘッド・チェックボックスのチェックを外してください。

次に、Undiscovered ReworkからWork Remainingにフロー・ツールを使って線を引きます。そのためには、まずUndiscovered Reworkをクリックして、それからシフト・キーを押下します。そして、シフト・キーを押下したままにして、右に真っ直ぐに動かしてクリックします。それから、下に真っ直ぐと線を引いて右クリックをします。それから、シフト・キーを押下したまま、左に動かして線を引きクリックします。シフト・キーは押下したまま上に動かしてクリックします。シフト・キーはそのままにして、左に動かし、最後に、シフト・キーを放して、"残っている作業"をクリックします。rework discovery rateという名前をバルブに付けます。すると次の様になります。



Work Accomplishedからスタートする最後のパイプを作成するためには、フロー・ツールを使用してください。既存のパイプが回り込んでいるのと同じ水平位置まで右に移動し、シフト・キーを下げたままクリックします。既存のパイプを底の位置まで移動して、シフト・キーを下げたままクリックします。最後にシフト・キーを放し、rework discovery rateのバルブをクリックします。

適宜、スケッチ移動・ツールを選択し、それらを適切に配列するために調整する必要があるかも知れません。

注意事項として、Work AccomplishedからWork Remainingにパイプを描くとき、Vensimはストックからフローへの流れを探知して、因果関係の方向を逆にして矢印の頭を取り除きます。このようにして、rework discoveryで始め、それから矢印ツールのオプションダイアログで矢頭のチェックを外すのと同じ結果になります。

3.4.2 エラーに関する処理の統合

ダイヤグラムを変更するとともに、次の様な式の変更や追加が必要になります。

rework discovery rate = Undiscovered Rework/time to detect errors

Units: Drawing/Month

time to detect errors = 3

Units: Month

time to detect errorsの値は、エラーにはどの様なタイプがあるかということ、そしてそれを誰が発見するのかということを考慮して決める必要があります。これは実際のところ、プロジェクト全体を通してみると定数ではないと思われます。これについては後でお話します。ここでは、平均2～3か月とするのが合理的です。

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

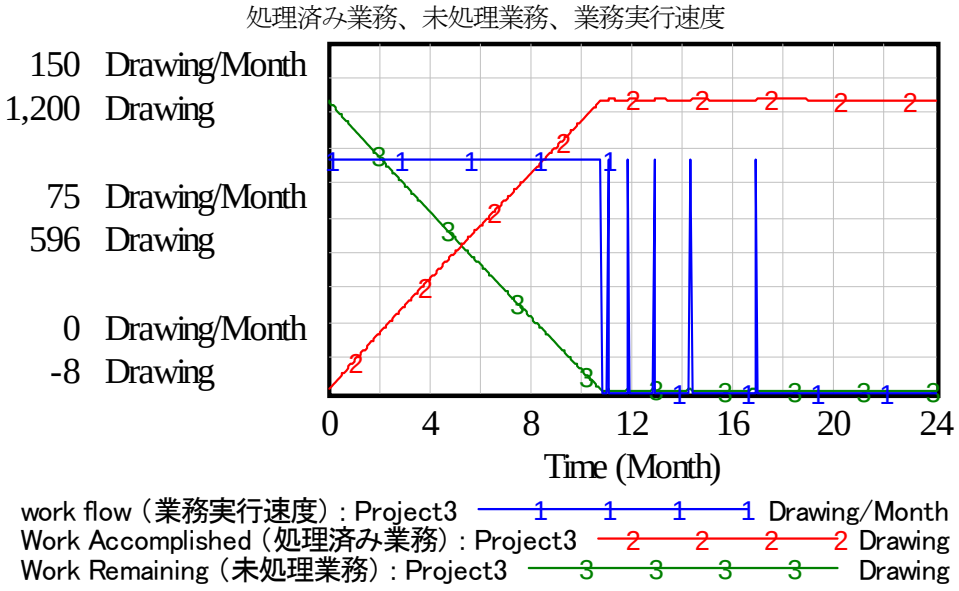
WORK QUALITY = 0.9

Units: Dmnl

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

Units: Drawing

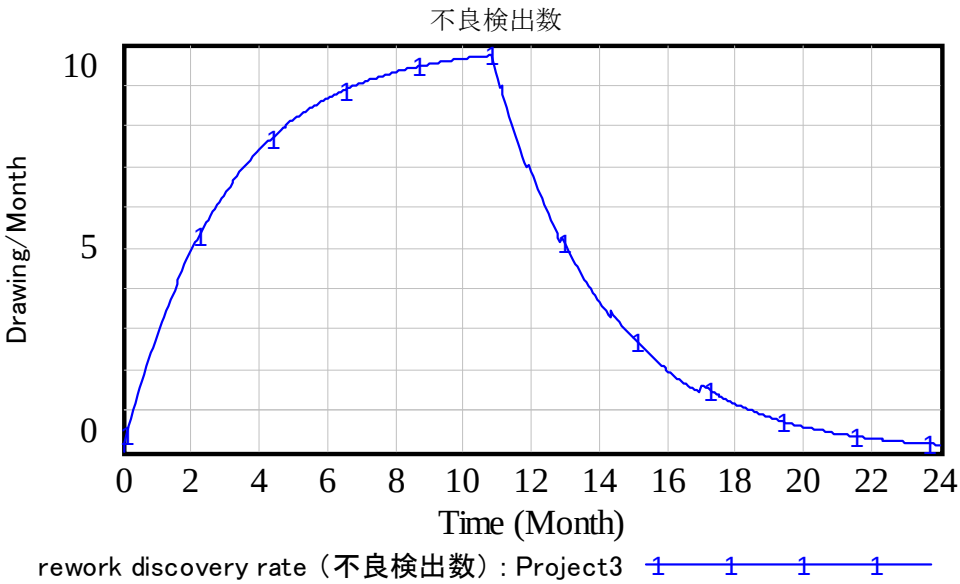
これらを付け加えると振る舞いは次の様になります。



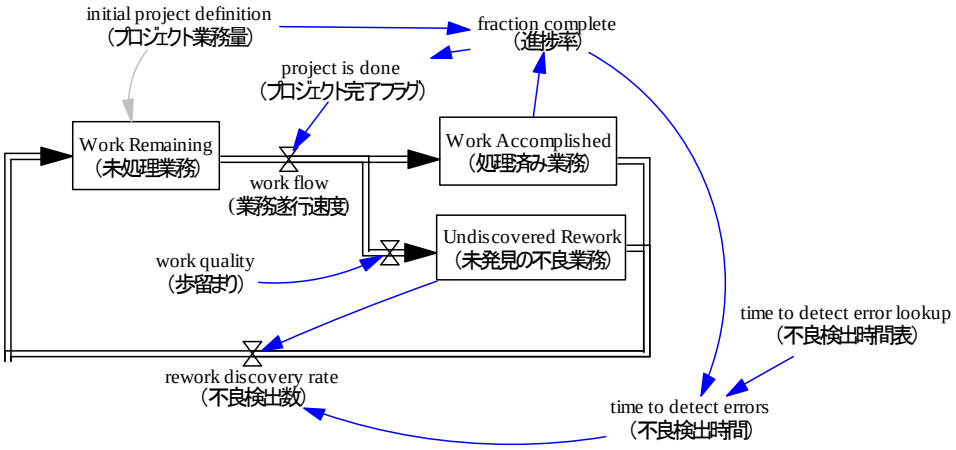
ここで2つのことに気づきます。1つめは、すべての仕事が正確に行われるとは限らないので、プロジェクトは少し長くかるということです。2つめは、プロジェクト終了の後、未発見のリワークが発覚して、何回もプロジェクトが再開されていることです。

3.5 リワークの発見(project4.mdl)

プロジェクトの終了時点でUndiscovered Reworkがピークに達しているので、rework discovery rateもまたその時点でピークとなります。



Undiscoveed Reworkは、本質的に観察によって発見できるものではないので、プロジェクトの最終段階ではリワークの発見量は大きく増加する傾向があります。それはパズルで最後のピースを置く際にピースの置き直しが多く発生することと非常に似ています。というのは、最後になって、それまでの見逃しや間違いが判明するからです。例えば、設備を導入する際に、段取りしていた廊下の大きさが間違っていたり、換気用のシャフトの寸法が間違っていたりするといったことが起こるのです。



これらの概念を表現するために、time to detect errorsを定義します。この変数はプロジェクトの完成状態に依存します。ここで、更にもう一つの変数fraction completeを加えます。そして、新しい変数を使って、project is doneの式を修正します。ここで変更され

る式は次の様になります。

fraction complete = Work Accomplished/initial project definition

Units: Dmnl

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

Units: Dmnl

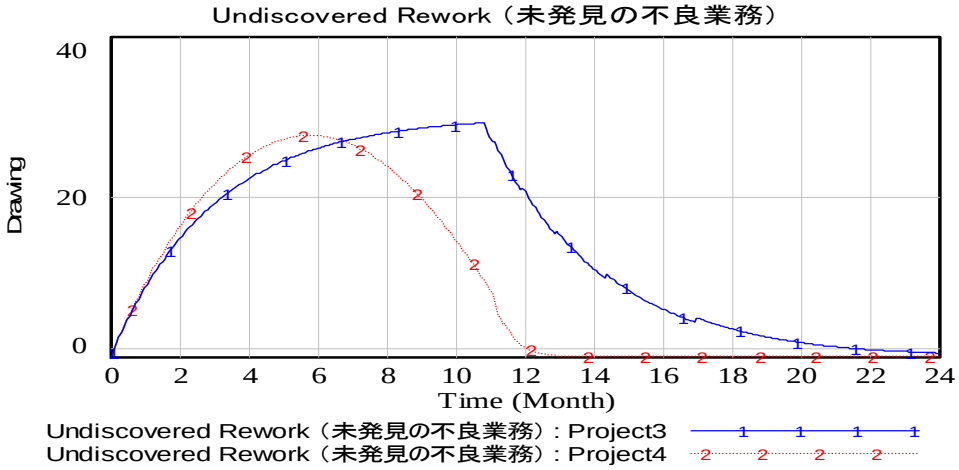
time to detect errors = time to detect error lookup(fraction complete)

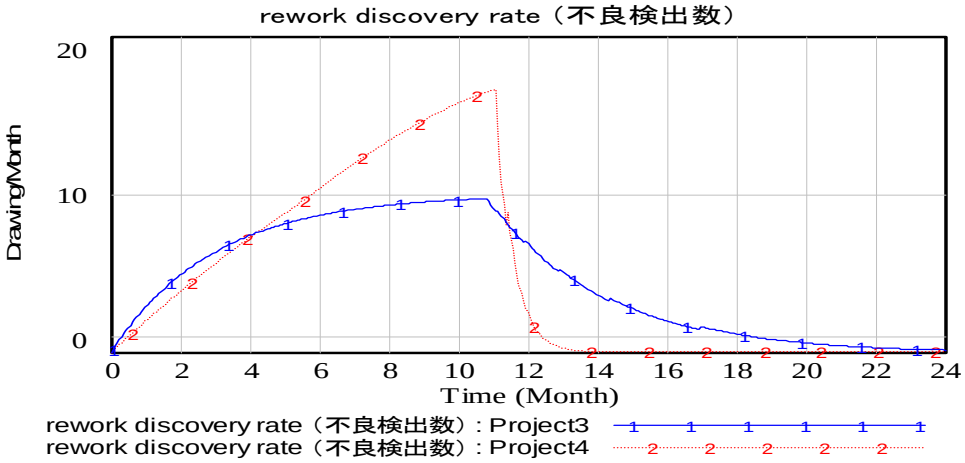
Units: Month

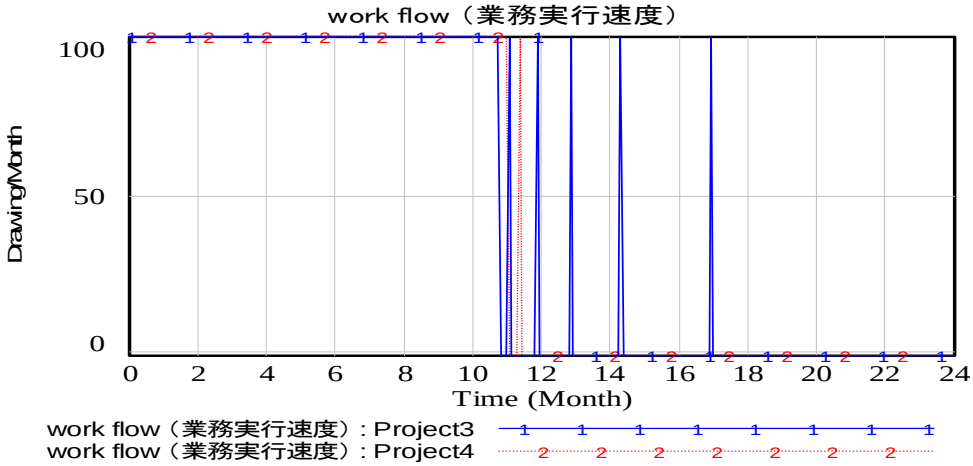
time to detect error lookup ((0,5),(0.5,3),(1,0.5))

Units: Month

プロジェクトが終了に近づくにつれて、充分に小さな値になる様にtime to detect errorsを変更するために、fraction completeをパラメータにして表関数time to detect errors lookupを使います。シミュレーション結果は次の様になります。





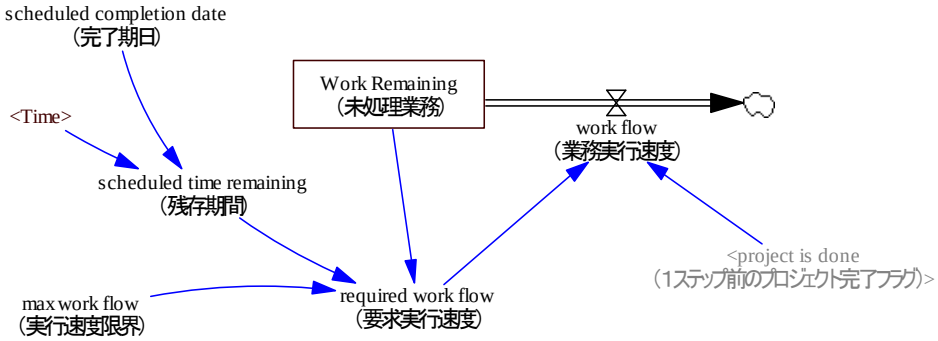


プロジェクトの終了はわずかに遅くなりますが、完成後のリワークがほとんど発生しなくなったことが分かります

シミュレーションでは、time to error detectの計算において、time to error detect lookupに関するエラーメッセージが出ます。これは、行われた仕事の量が行うべきオリジナルの仕事量をわずかに超えることに対して起こるものです。これは不連続的にプロジェクトを止めたことが原因と考えられます。これはコンセプトとしてなんら問題はありませんので、今のところそのままにしておきます。

3.6 スケジュール(project5.mdl)

これまでのところ、work flowは定数として表しています。だから、スケジュールからのフィードバックもありません。プロジェクト管理の目的はプロジェクトをスケジュールどおりに進めることです。そのためにスケジュールに注意を払い、スケジュールに合わせてリソースを調節することが必要です。完成予定日があるならば、それに基づいて残された日数を計算することができます。また、残りの仕事の量は分かっているので、スケジュールに間に合わせるために、仕事をどれだけ早めなければならないかということを決めることができます。分かりやすくするために、今後は変化した部分の構造だけを表す様にします。



式は次の様になります。

required work flow = MIN(max work flow,XIDZ(Work Remaining,scheduled time remaining,max work flow))

Units: Drawing/Month

ここでは、MIN(minimum)関数を使って、要求されたワークフローを最大のワークフローよりも小さくなる様に制約します。XIDZ(X if Divide by Zero)関数は、もしscheduled time remainingがゼロでなければ、Work Remainingをscheduled time remainingで割ります。0で割ることは意味がなく、Vensimはエラーとなるので、今後はこの様な場面ではこの関数を使うことが重要になります。

scheduled time remaining = MAX(0,scheduled completion date - Time)

Units: Month

scheduled completion date = 10

Units: Month

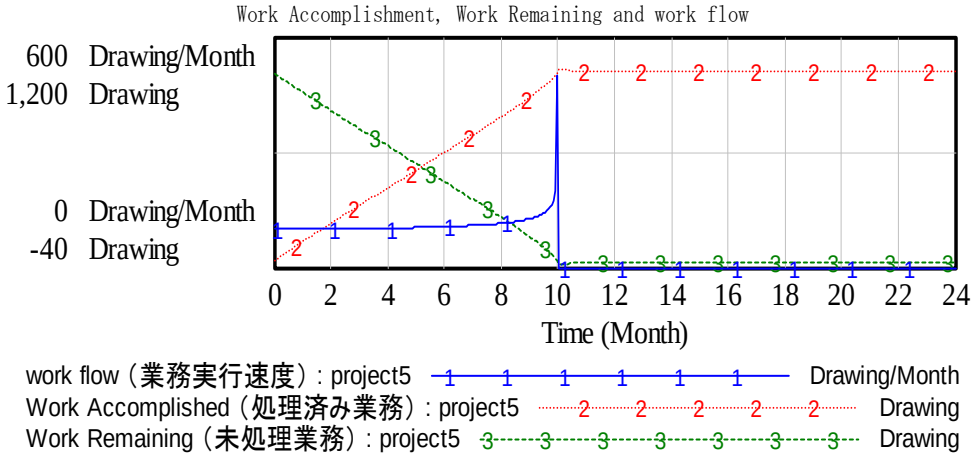
max work flow = 500

Units: Drawing/Month

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

Units: Drawing/Month

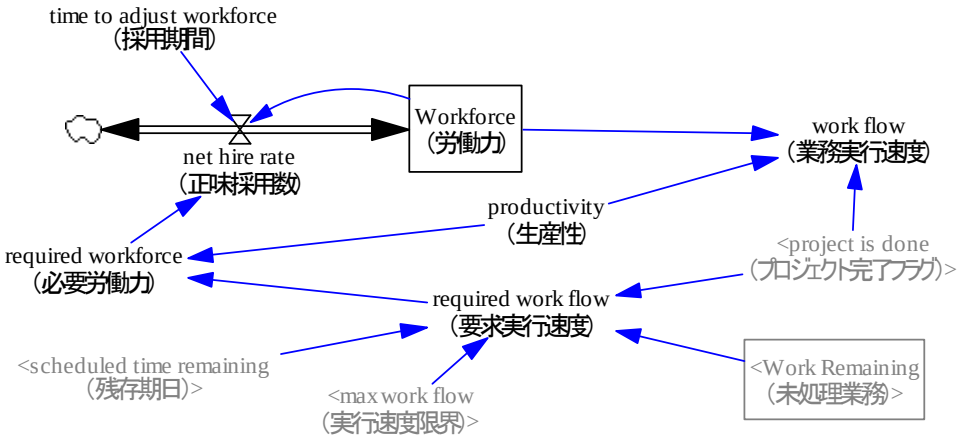
シミュレーション結果は次の様になります。



プロジェクトは時間通りに完了します。work flowはゆっくりと上昇し、次に、時間通りにプロジェクトを終了するために終了直前で急激な上昇を示します。

3.7 労働力と雇用(project6.mdl)

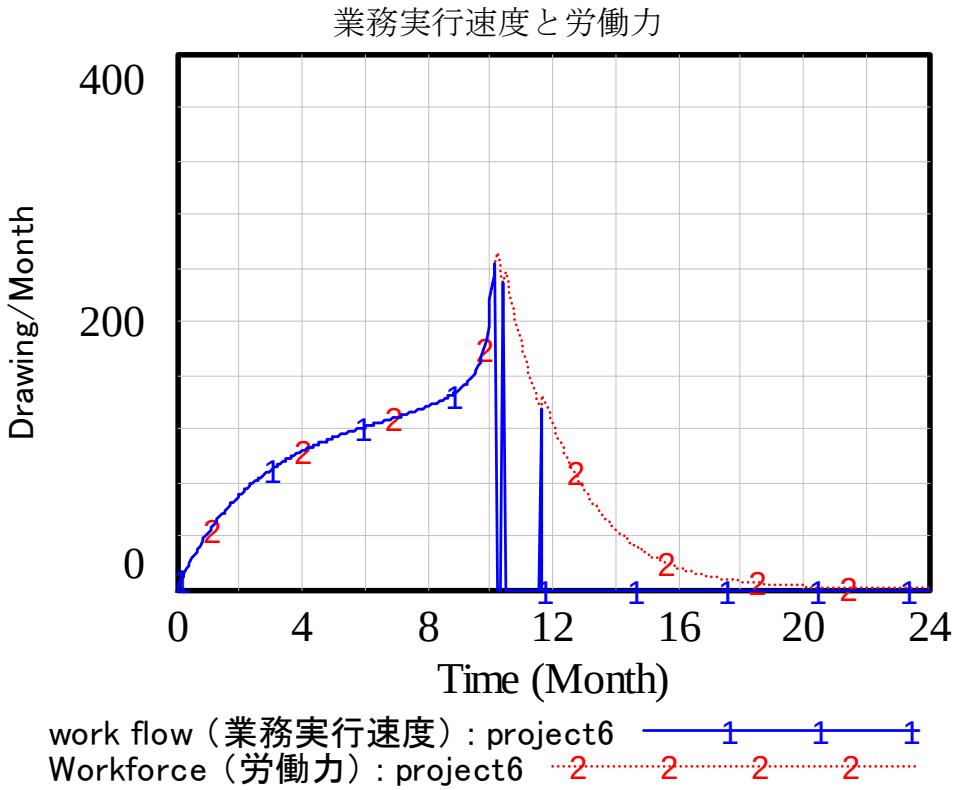
今までは、制約として、仕事を完了するために必要なすべてのことをwork flowとして扱って来ました。しかしながら、実際には仕事を遂行するためにはリソースが必要です。このモデルでは、作業レベル(effort)に注目しています。そしてリソースは人です。これらの人は、仕事のために新たに雇われるか組織内的にアサインされるかにかかわらず、仕事を終わらせるために必要となります。 労働力最もシンプルな定式化は労働力-在庫モデルの中で使用したものと同じです。この場合、望まれる生産量は時間どおりにプロジェクトを完了するのに必要なワークフローということになります。



required workflowに関する式は、プロジェクトが完了すればゼロになる様に変更します。新しい式は次の様になります。

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

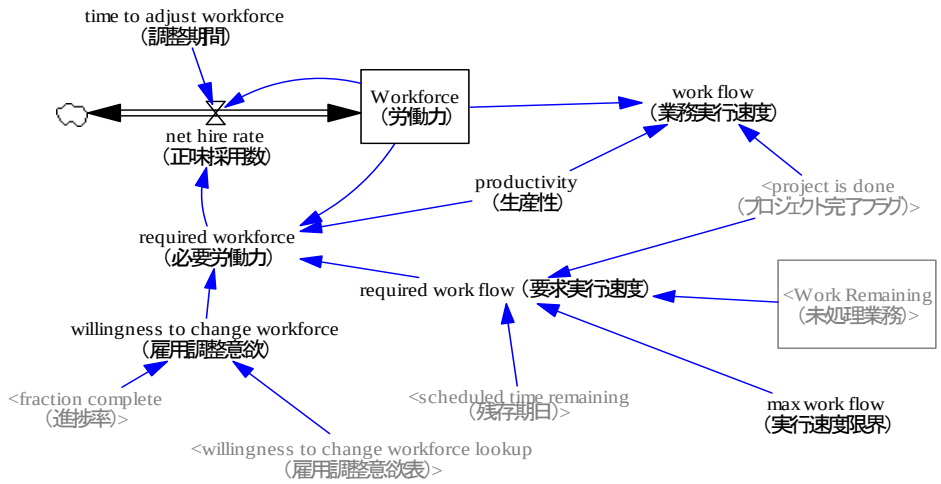
プロジェクトは、作業する人が誰もいなくても始まります。また、人の募集は比較的速くできます。この様な見方は、必要労働量とその募集に関する最も分かりやすい見方です。始めと終わりで作業レベル(effort)を緩やかに加減することや、作業の集中度合いを計画することも可能でしょう。このモデルのシミュレーション結果は次の様になります。



Workforceは、100人まで迅速に増やし、その後はゆっくりと増やし、最後にプロジェクトが終了する近傍で急激に増加します。そして、プロジェクトの終了の後にプロジェクト活動が再開します。これら2つの非現実的な振る舞いについては両方とも対処が必要です。

3.8 労働力調整意欲(project7.mdl)

まず、プロジェクトの終了が近づくにつれて、より多くの人を投入することは実際にはあまりありそうにない話です。プロジェクトが進行するにつれて、プロジェクト・チームは役割や活動の観点で安定する傾向があります。また、プロジェクトが80%完成した後に要員を増やすということは大抵の場合不適切です。この様な事情をwillingness to change workforce(労働力調整意欲)という変数を用いてこれらのダイナミックスを捉えます。

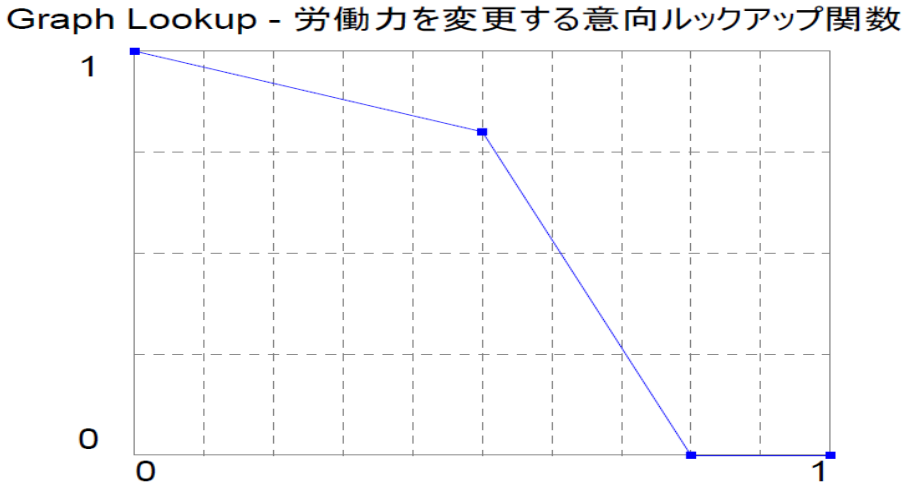


今回変更した式では次のとおりです。

```
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
```

この方程式は労働力が必要量より大きいときは労働力を削減しようとしますが、その際に、プロジェクトの末期における労働力の増加は許さない様になっています。

```
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

3.8.1 プロジェクトの再開

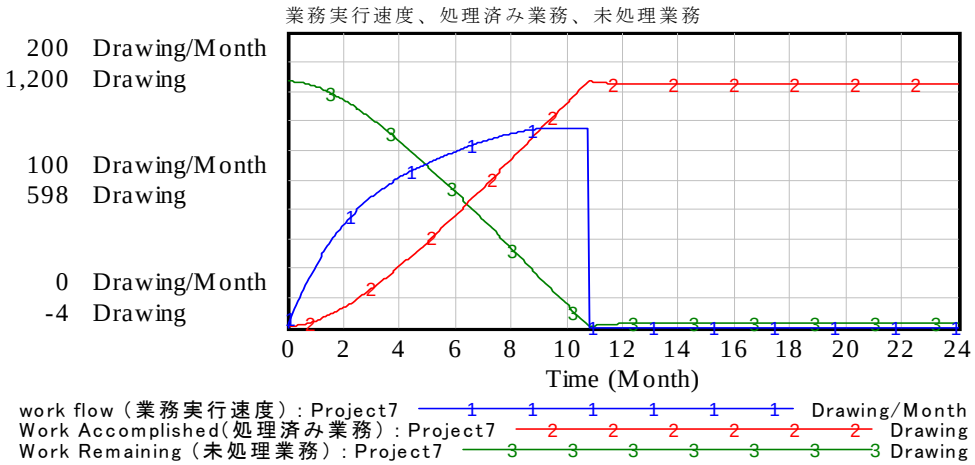
見たところは成功して終わったプロジェクトに問題が発見されたために、再開することはありそうなことです。しかしながら、一般的にその様なことが発生する訳ではなく、大きな問題が発見されたときだけです。

ここで考えているモデルは、もっぱらプロジェクが完成する前のダイナミックスに関心があるので、プロジェクトの再開の問題はそれほど重大ではありません。1つのプロジェクトの中に複数のステージを含むプロジェクトで、2つ以上の活動が行われ、かつ活動間でリソースをシェアするときは、タスクの再開という意味で重要な問題になってきます。これをモデリングする方法は、8章で議論するサーモスタット問題の取り扱いとほとんど同じです。プロジェクトが一旦完了したと宣言された場合、余程事態が悪くならない限り、プロジェクトを再開する様なことはしません。(これをヒステリシスと言います。)

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

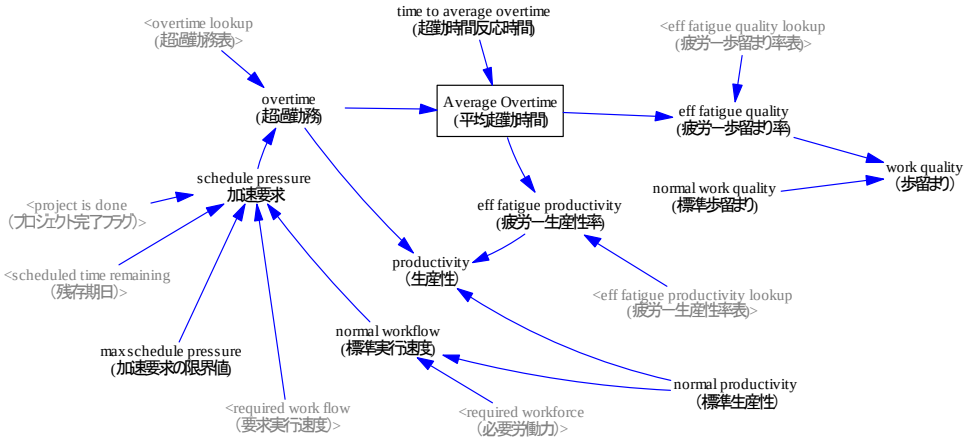
3.8.2 結果として観察される振る舞い

このプロジェクトの完了は少し遅くなっています。そして活動はプロジェクトの終了時点に向けてフラットになって行きます。プロジェクトが完了した後、発見されたリワークが少し発生しています。これはプロジェクト再開を引き起こすほど大きくならず、プロジェクト再開の宣言はなされません。


業務実行速度、処理済み業務、未処理業務
200 Drawing/Month
1,200 Drawing
100 Drawing/Month
598 Drawing
0 Drawing/Month
-4 Drawing
0 2 4 6 8 10 12 14 16 18 20 22 24
Time (Month)
work flow（業務実行速度）: Project7 Drawing/Month
Work Accomplished(処理済み業務）: Project7 Drawing
Work Remaining（未処理業務）: Project7 Drawing

3.9 スケジュールによるプレッシャ(project8.mdl)

プロジェクトの終結時点付近で労働力を上手く凍結してしまうことで、スケジュールに合わせようとするための過剰な活動の増加をなくすることができました。プロジェクトの最終段階において、通常のスピードで働き続け、仕事が完了したときにプロジェクトが完了する様になります。しかし、この際に要員数を一定とするのは妥当ですが、作業レベルまで一定ということは考えられません。



ここで、 overtime(超過勤務)につながるschedule pressure(加速要求)という変数を導入します。そして、平均残業時間が大きいと全体的に残業による疲れが蓄積してきていると考えられるため、Average Overtimeを残業の結果発生する疲れを表す尺度として用いることができます。疲れは、生産性と作業の質を低下させます。

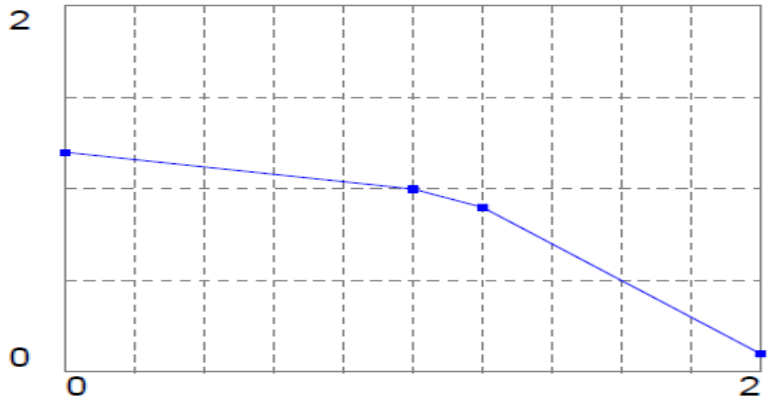
変更した式は次のとおりです。

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

この式は、Average Overtime=SMOOTH(overtime, time to average overtime)と同じです。一般的には、SMOOTHの様な、ダイナミックス関数は避けねばなりません。何故ならば振る舞いがおかしい場合、その原因の追究を困難にするからです。ダイナミックス関数は式の見通しが良いときだけ使う様にすることが大切です。

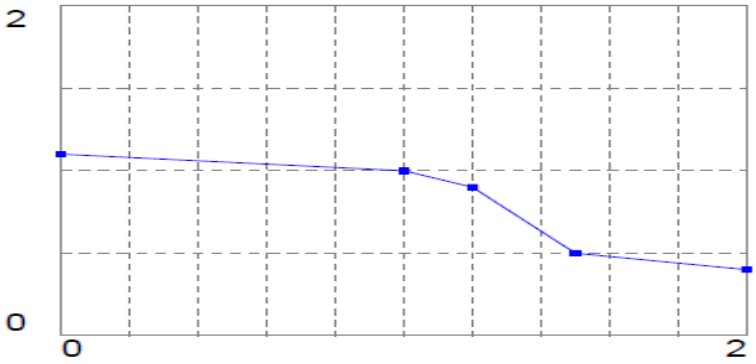
```
eff fatigue productivity = eff fatigue productivity lookup(Average Overtime)
Units: Dmnl
```

Graph Lookup - 疲労の生産性へ影響ルックアップテーブル



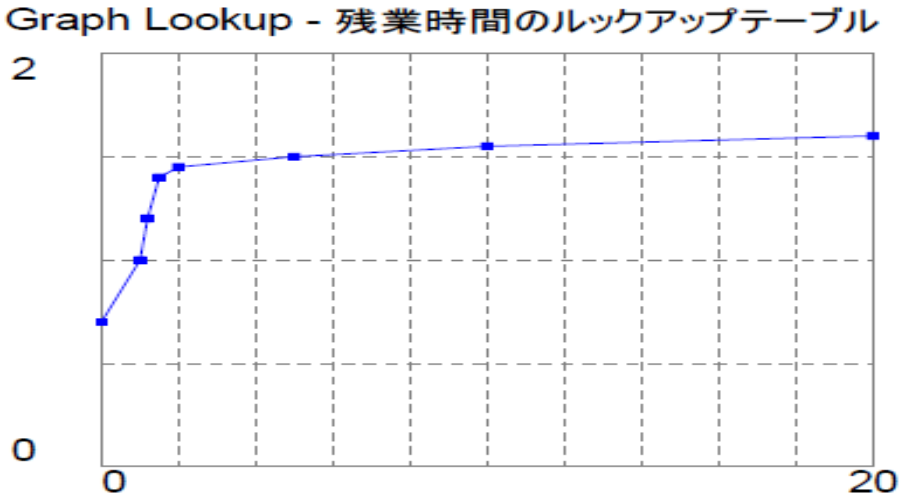
eff fatigue productivity lookup((0,1.2),(1,1),(1.2,0.9),(2,0.1))
Units: Dmnl
eff fatigue quality = eff fatigue quality lookup(Average Overtime)
Units: Dmnl

Graph Lookup - 疲労の作業の質への影響ルックアップテーブル

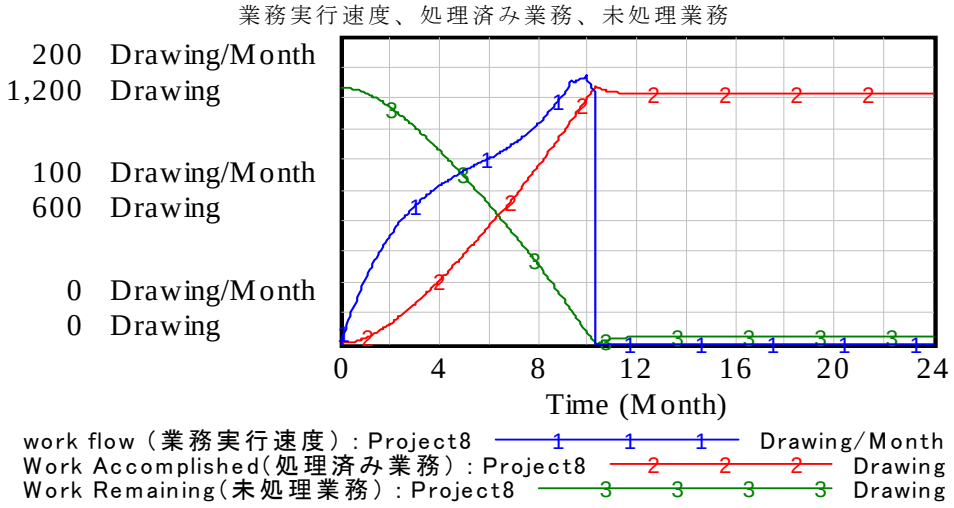


eff fatigue quality lookup((0,1.1),(1,1),(1.2,0.9),(1.5,0.5), (2,0.4))
Units: Dmnl
max schedule pressure = 5
Units: Dmnl
normal productivity = 1
Units: Drawing/Person/Month
normal work quality = 0.9
Units: Dmnl
normal workflow = MIN(max work flow, normal productivity * required workforce)
Units: Drawing/Month
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

これらの式では、productivityの代わりにnormal productivityを使用する様に変更しました。もしその様に変更しなければ、超過勤務が織り込まれた高い生産性で計算されてしまい、この高いアウトプット量が、新規に人を投入する際の基準とみなされてしまいます。プロジェクトによっては、このことは実際には現実的なことかもしれません。しかしながら、この場合、生産性ではなく認識された生産性の平均を使用する必要があります。何故ならば、生産性は直接観察できないからです。



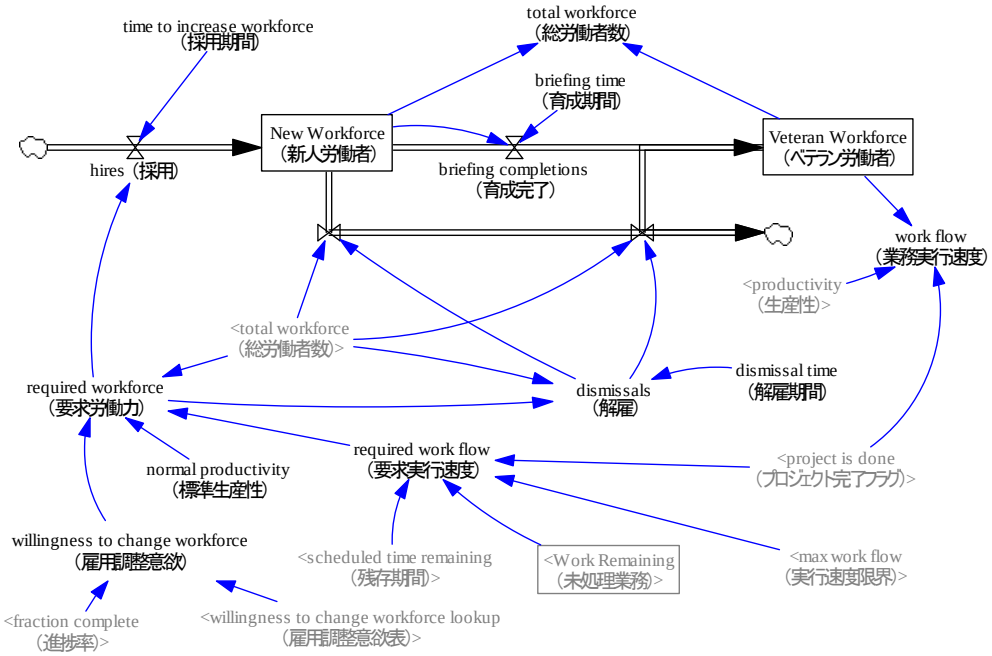
eff fatigue qovertime = overtime lookup(schedule pressure)
Units: Dmnl
overtime lookup((0,0.7),(1,1),(1.2,1.2),(1.5,1.4),(2,1.45),(5,1.5))
Units: Dmnl
productivity = normal productivity * overtime * eff fatigue productivity
Units: Dmnl
schedule pressure = IF THEN ELSE(scheduled time remaining<= 0 :AND:
 :NOT: project is done,max schedule pressure,
 ZIDZ(required work flow,normal workflow))
Units: Dmnl
シミュレーションの開始時にはnormal workflowは0なので、ZIDZ関数を用います。一旦スケジュール遅れが生じた場合、schedule pressureは完成期日まで、その最大値まで増加します。
time to average overtime = 2
Units: Month
work quality = normal work quality * eff fatigue quality
Units: Dmnl
これらの変更で、振る舞いは次の様になります。



終了に向けた活動のピークは超過勤務で遂行されます。終了付近では作業の質は下がります。また、少しですが、プロジェクト完了後に発見されるリワークが観察されます。

3.10 労働力の調整(project9.mdl)

上記のシミュレーションでは、プロジェクト終了後しばらくの間はプロジェクトに割り当てられた人が残ることが分かります。　勿論、段階的に削減する努力は意味あることですが、それにしてもあまりに急激にプロジェクトの完了が発生しそうです。これに対しては、人員の獲得と解雇では時定数が異なると仮定しておくのが適切であることが示唆されます。次に考察上重要なことはプロジェクトに投入する人の準備です。いままで、人は誰でも直ちに仕事ができると考えてきました。プロジェクトに投入される人は熟練していると仮定しますが、責任を果せるスピードになるまで、トレーニングが必要になります。

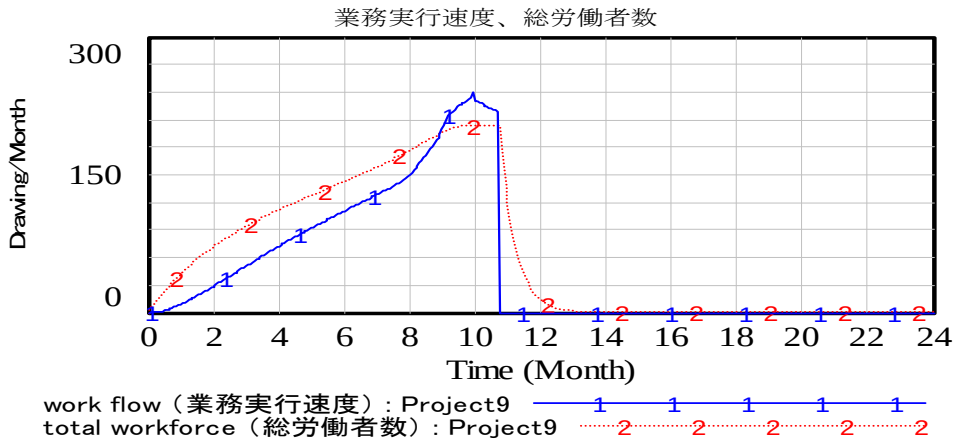


労働力は、2つに分けられています。採用されて入ってきた人は、まずNew Workforceの集合に入ると考えます。その後に、十分なトレーニングを経てVeteran Workforcになって行きます。そして、ベテランだけが実際にWork flow(業務遂行速度)に寄与するものとします。従って、New Workforceを持ちすぎると、負担が大きくなってしまいます。他方、必要な労働力を計算する際には、total workforceに対して比較がなされます。hiresはNew Workforceにのみ流入していますが、hiresは新しい労働力およびベテランの労働力の両方から比例配分してなされるとします。hiresに要する時間は、労働力を増加させる時間より短い時間で準備が整うと仮定します。

変更された式は次のとおりです。

```
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
関数ZIDZはtotal workforceが0のとき、0で割ることを防ぐものです。
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
　また、このモデルを実行すると、次に示す振る舞いが得られます。



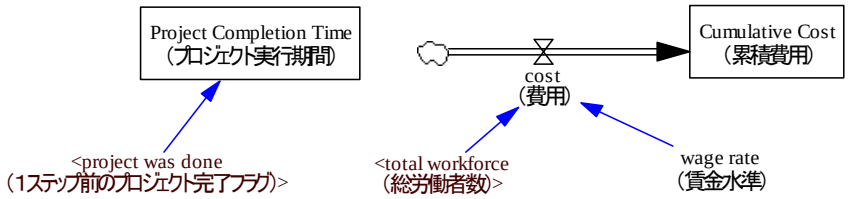
　ここで重要なことは、total work forceからwork flowが逸脱していることと、より多くの人が瞬間的に必要になっているということです。

3.11 ポリシーの検証(project.mdl)

一連の小さなステップを積み上げることによって、興味深いプロジェクト・モデルを開発することができました。このモデルを使って、とりうるポリシーに関するアイディアの検証をしてみましょう。検証のために、累積費用とプロジェクト実行期間を知るための式を付け加えます。その後、雇用に関するポリシーのオプションをコンピュータ・シミュレーションしてみます。そしてパフォーマンスを改善できるかどうかを確かめます。

3.11.1 会計情報を集計するための式

モデル全体をサマリする様な測定項目を付け加えることは多くの場合有用です。この例の場合、最も興味深い測定項目は累積費用とプロジェクト実行期間です。



式は次の様になります。

```
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
```

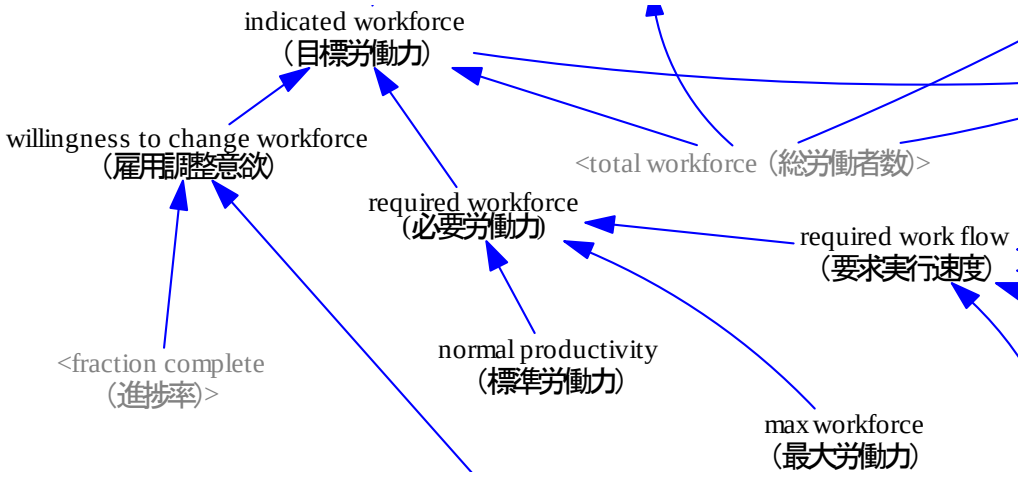
関数SAMPLE IF TRUEはプロジェクト完成済フラグが真になるまでTimeを戻し、それ以外はパラメータで与えられた値を戻します。この定式化により、成果が低品質であることで生じるプロジェクトの再開を捉えることができます。

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

WAGE RATEには利益を含んでいるものとします。

3.11.2 労働力の上限

労働力に関するポリシーを試す実験として、労働力に上限を設けることを考えてみます。仮に完成させるべき図面が1,000あり生産性を1とすれば、単純な計算で、100人で10か月かけて仕事を終了することができることが分かります。もし人員が2倍いたとしたら、恐らく、もっと早くできるに違いありません。必要な労働力あるいは目標とすべき労働力といったものを改めて明確にします。しかし、既に現在の方程式は複雑過ぎます。そこで、分かりやすくするために、required workforce(必要労働力)をindicated workforce(目標労働力)と名前を変えることにします。そしてそのうえで、改めてrequired workforceと呼ばれる新しい変数を付け加えました。



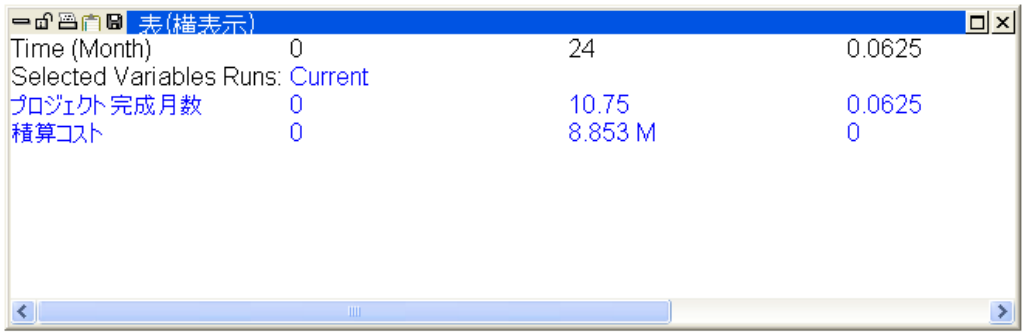
indicated workforceは、required workforceが使われていたところに使われることになります。そして、required workforceに対しては別途計算方法を定義します。この際に、required workforceの名前を変更したとき、Vensimはそれが使われていた式の中の変数名も自動的に変更します。このようにして、新しい式は次の様になります。

```
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
```

労働力に関しては無制限とするというポリシーを試すために、最大の労働力には初期値として非常に大きな値をセットしています。これらの変更をモデルに加えてシミュレーションしたならば、同じ振る舞いが生成されます。

3.11.3 最終結果の表示

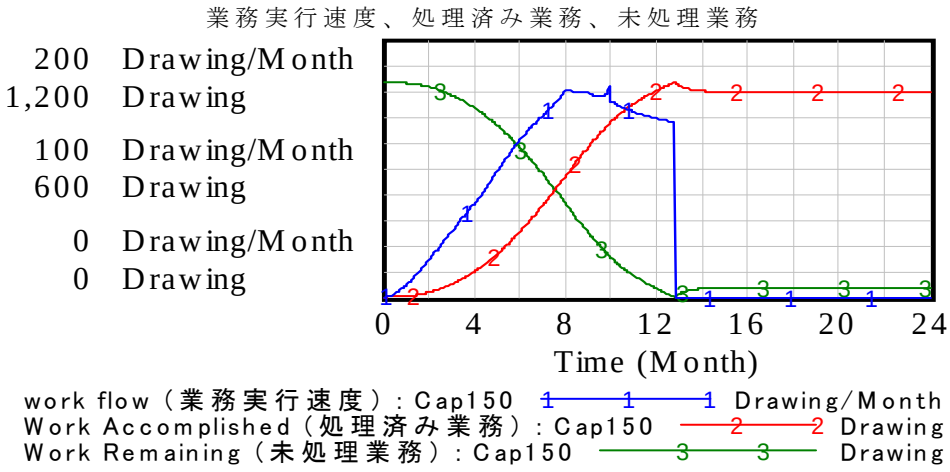
関心を払うべき数字として、累積費用とプロジェクト実行期間という2種類の尺度を設定しました。最終的に値がどの様になったかを知るために“表ツール”を使います。“制御パネル”の“時間軸”タブをセットして、とても小さな範囲だけを使う様にセットします。（リファレンスマニュアルの12章参照）そして、“表ツール”をクリックすると次の様な表が作成されます。



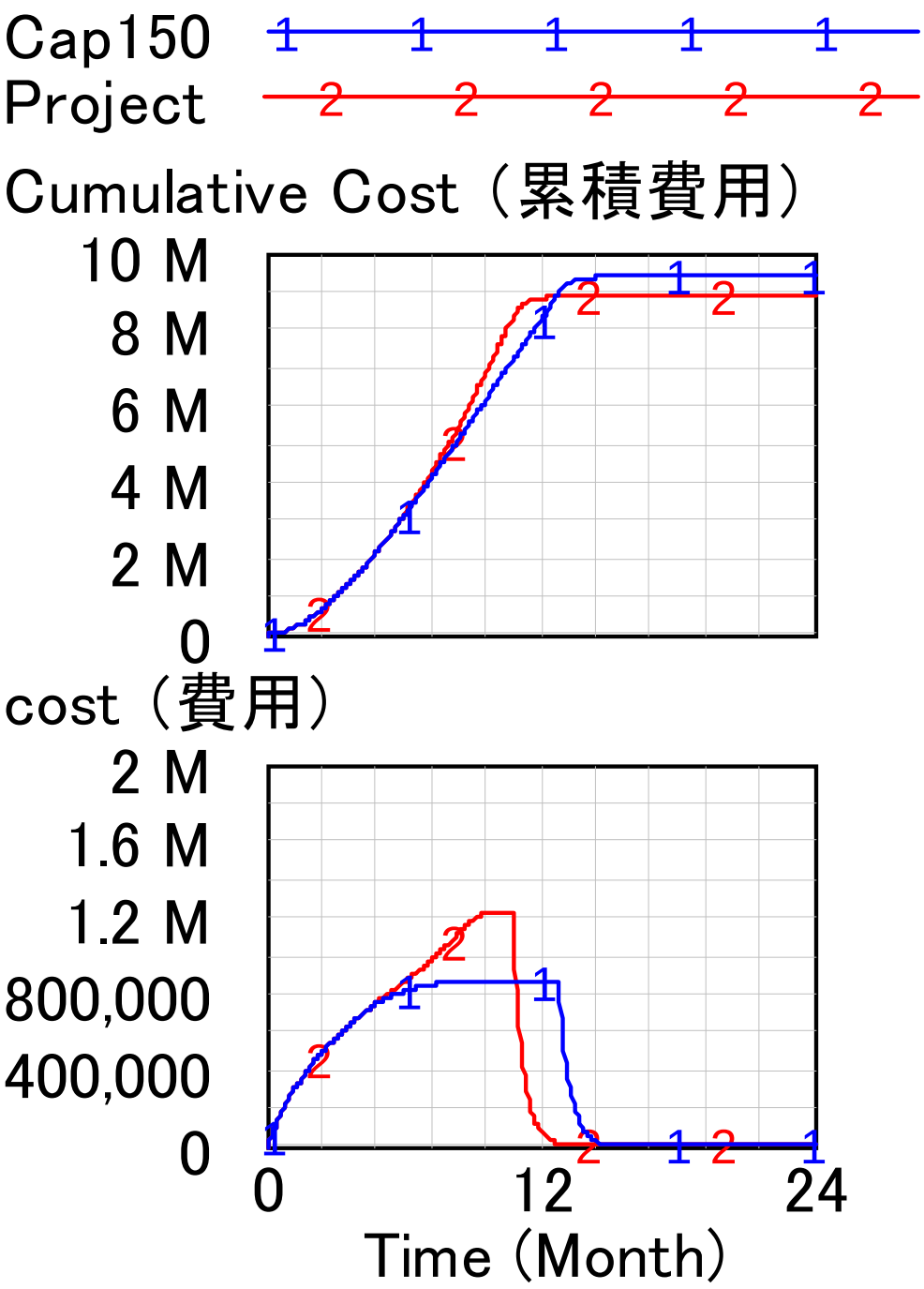
もし、2つ以上の実行をロードしていたら、そのモデルでは使われていない変数については空行になります。プロジェクトコストは、8.85百万ドルになり、プロジェクトは10.75ヶ月目に完了します。

3.11.4 労働力の上限

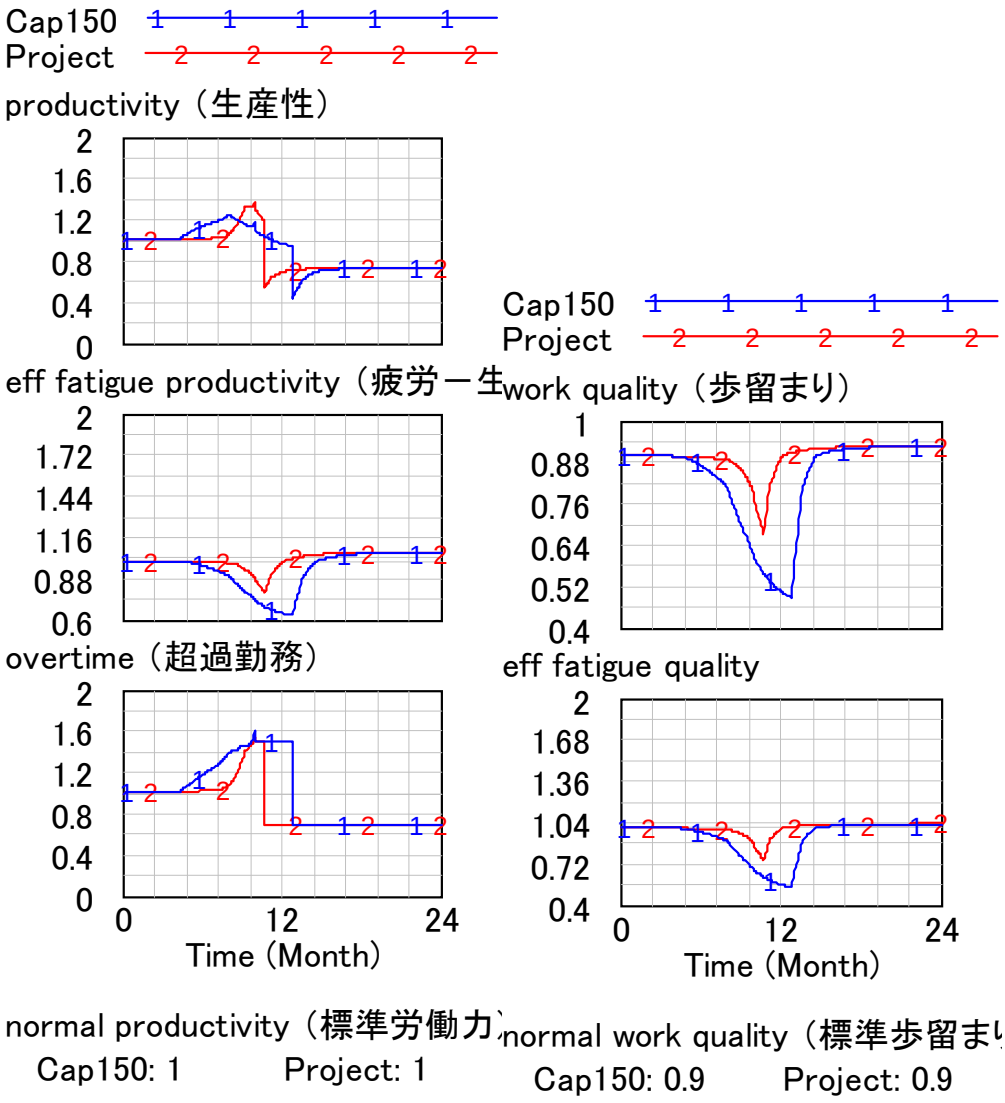
労働力の上限を150人として実験的にシミュレーションを行います。すると、次の様な結果になります。



労働力に上限を与えたことで、プロジェクト実行期間は少し伸びます。しかし、コストに関しては少々驚くべき結果が出ています。累積コストは労働力に上限を置いた方が高くなります。なぜでしょうか？累積コストを選択して、直接原因グラフを使います。(制御パネルにおいて時間軸タブを開いてリセットすることを忘れない様にしてください。) もし、正しい実行結果が出ないときは、データセットタブで正しいデータセットをロードしてください。すると、累積コストに関して直接原因グラフをクリックすれば次の様な結果が得られます。



確かに、シーリングがない場合に比べて月当たりの人件費はCAP150の方が高くなりません。しかし、CAP150ではシーリングがない場合よりも、より長い間コストが継続してしまいます。このために、積算コストが高くなってしまうのです。　次に、“生産性”を選択して、“直接原因グラフをクリックしてください。



労働力の上限を150人とすると、生産性はあまり大きくは上昇しません。スケジュールからのプレッシャが早い時期からかかり、労働力を疲弊させ続けます。それに伴って、作業の質もまた大きく低下します。それに対して、ベースとなるモデルでは、皆が疲労困憊して燃え尽き症候群に陥る前に、アクティビティをどんどん終了させてプロジェクトが完了してしまうのです。

3.12 要約

この章では、小さな部品を、既存のモデルに付け加えて行くことで、ステップ・バイ・ステップでモデルを構築して行くプロセスを経験しました。この方法の良いところは、常に動作するモデルがあるということです。良くない点は、木を見て森を見失うことにもなり得るという点です。また、いつ付け加えるのを止めるかということも難しい問題です。大きな視野を持っていない場合、理解を深めないまま、どんどん詳細を付け加えて行ってしまいます。小さなステップを積み上げることは強力なアプローチです、しかし、ステップの積み上げが、望む方向に向かっているかどうかを確かめるために時々セルフチェックすることが必要です。

第４章

学問分野の発展

4.1　はじめに

新製品や新技術が市場へリリースされるとき、種々様々な拡がり方を見せます。一時の流行もありますし、悪化する災害の様な酷い失敗もありますし、予想外のヒット、例外的にとしか言えないことは悲しいことですが､本当に長期的な大成功を示すこともあります。個々の導入された新製品や新技術はユニークでも、それらが示すダイナミックな振る舞いには共通点があります。

ここでは、システムダイナミックスの分野を研究するという学問分野の発展を具体的に考え、その中でVensimの様な技術の役割を理解してみましょう。まず、問題の背景から始めましょう。そして、何が起こっているか、また、どの様な施策を採り得るのかという多くの仮説を先に列挙します。そして、その後、これらの仮説をテストするためにモデルを構築します。

本章の中で使用するモデル構築プロセスは、理論を明瞭に表現し形式化したうえで、一つの統合化されたモデルにそれらを組み入れようとするものです。しかし、多くの理論は上手く形式化されておらず、内部に矛盾を抱えているため、このプロセスは通常順調には進みません。内部的に一貫しない理論を形式化しようとすると問題が発生します。しかしながら、それらが上手く意味をなす様に、矛盾点を解決することは困難かもしれません。このモデル構成プロセスに伴うもう一つの難点は､2つの理論が単一のフレームワークに必ずしも入れることができるとは限らないということです。ある場合には、異なる理論のための異なるモデルを開発し次に、振る舞い、実現性の点検とデータに基づいてモデルを比較する方が簡単です。

4.2 予備知識

1956年に、ジェイW.フォレスターが、経済と管理問題に関する研究にフィードバックとコントロールの法則を適用し始めました。フォレスターは、研究分野毎にばらばらに問題に集中しても、本当に優れた成果に結び付くレバレッジが効く解決策を得られないのではないかと感じました。そこで、彼は、システムダイナミックスの分野を開拓したのです。

1956年以来、この分野は発展してきましたが、そのスピードは期待したよりははるかにゆっくりであると、研究を引き継いだ誰もが感じるものでした。

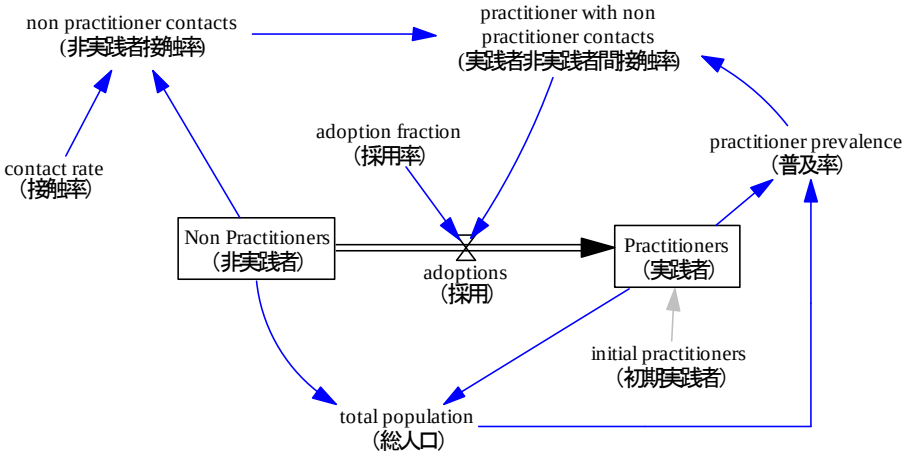
4.2.1 仮説

この分野の成長を見る際に、多くの仮説となりうる考えを書き出すことができます。

- 発展は指数的であるがゆっくりである。(本当のところ、事態は良好)
- 質の高い教授陣が不足している。(供給を増加させるべき)
- 充分な教科書がない。(訓練の標準化が必要)
- 単純なモデルであれば、常に広く流布できる。(様々な目的に適用できる様にすべき)
- モデルを理解するのを助けるために技術が必要(アクセスが容易になるように改善すべき)
- システマティックに思考するための再教育が必要(需要の喚起が必要)
- 大抵の人はモデルを構築することができない。(もともと潜在市場は小さい)
- より熟練した実践者が必要。(実践の質の向上が必要)
- 単に難しすぎる。(アイディアをよりシンプルにする)
- ニーズとスキルのミスマッチ (トレーニングの方向の見直しが必要)
- 成功例を公表する。(マーケティング活動を増やす)

4.3 基礎的な普及プロセス (sdgrow1.mdl)

どの仮説を考慮するかにかかわらず、どれだけの人々が、この技術を使用しているかについて考えてみる必要があります。数人の人々の仕事で始まり、次第に広がって行くというプロセスを踏みます。この非常にシンプルな考えを心に留めながら次のモデルからスタートしてみます。



このモデルには2つの「人々の集まり」があります。すなわち、PractitionersとNon Practitionersです（ここではできるだけニュートラルな言葉を使う様にします)。ここでは人々はすべて、会議に行き、ミーティングに参加し、通路で偶然にばたりと突き当たります。誰かが他人の中に、この技術についてのアイディアを注入するためには、取り込み（採用）のプロセス(process of adoption)が必要なので、Non Practitionersが Practitionersに遭遇する頻度が重要になってきます。従ってnon practitioner contactsを乗じることにします。

practitioner with non practitioner contractsとはPractitionersと、Non Practitioners間の接触を表します。この出会い結果、Non Practitionersが実践する様になる可能性があります。この様なことが起こる確率をAdoption fractionとします。

上記モデルの式は次のとおりです:

```
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
```

このモデルの人数は10百万人で始めます。これは、学者、およびこの種の仕事がそのために適切な、熟練した専門家の数から持ってきました。この数はここでの議論を行うため

には、外から与えられるものとして差し支えない問題です。そのことは、この章ではいちいち述べることはしません。

practitioner prevalence = Practitioners/total population

Units: Dmnl

practitioner with non practitioner contacts = non practitioner contacts * practitioner prevalence

Units: Person/Year

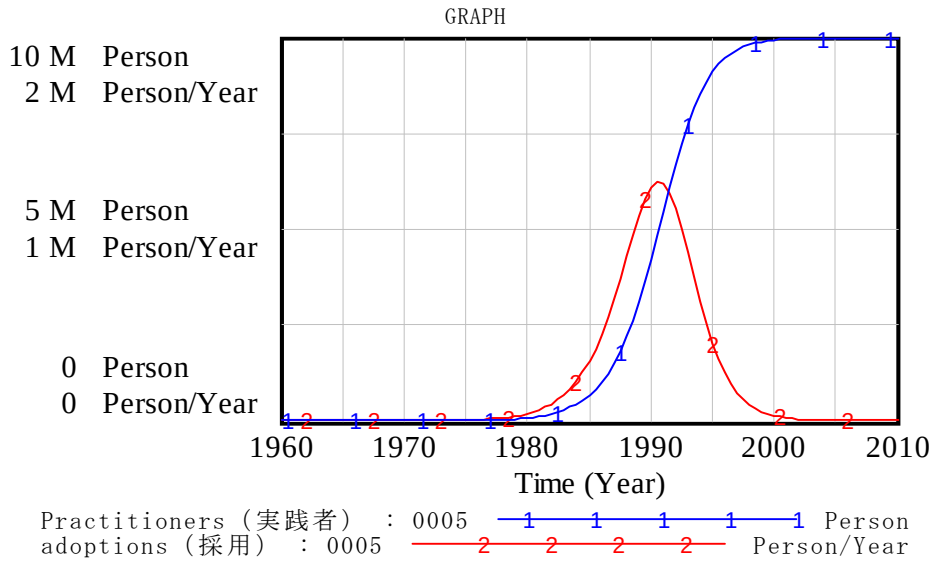
Practitioners = INTEG(adoptions, initial practitioners)

Units: Person

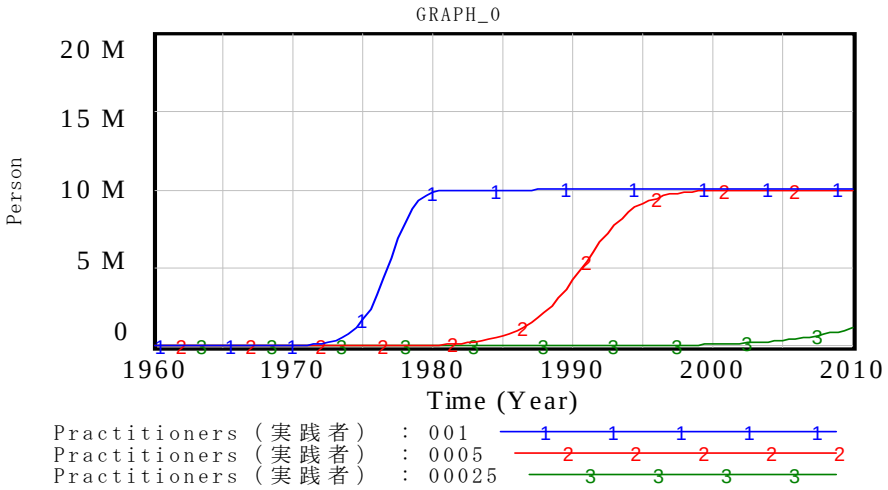
total population = Non Practitioners + Practitioners

Units: Person

モデルはTIME STEP=0.125.で、1960年から2010年まで実行されます。このモデルは次の振る舞いを生成します。



モデルの振る舞いは、与えられたパラメータ値に大変敏感です。例えば、採用率として0.01、0.005および.0025を与えれば次の様な結果を得ます。



0.01の様な大きな値では、Practitionersの数は急速に増大し浸します、すべての人がこの方法を採用するとともに飽和状態になります。他方、0.0025の値については、Practitionersの数は、このスケールではわずかに現れてくる程度です。行うべき面白い1つの実験は、0.001まで採用率を更に低下させて、飽和状態に至るまでにどれくらいの時間がかかるか確かめることです。

4.3.1 振る舞いに関する注釈

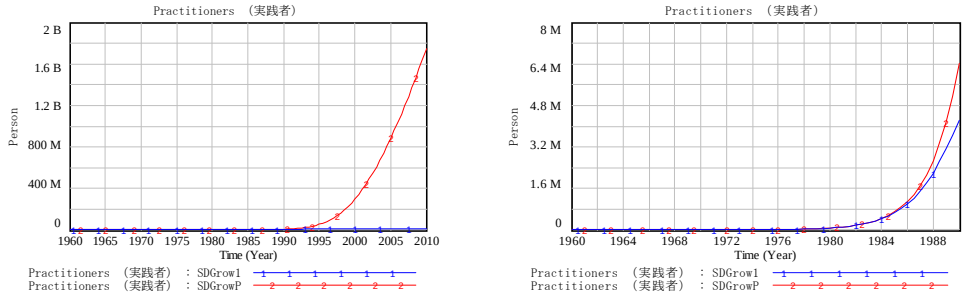
指数関数的成長の最も重要な特徴の1つは、長い期間、活動が外見上はほとんど目立たずに、そしてあるとき爆発が起こることです。このモデルでは、adoption(採用)のフローがPractitionersに向かい蓄えられると同時に、成長の源泉である、Non Practitionersのストックを消耗するので、爆発は阻止されます。もし、ストックNon Practitionersを定数で置き換えることによりモデル中のこのリンクを壊してしまった場合は、次の様になります。

Non Practitioners = 1E7

そして、真の指数関数的成長とするために次の様におきます。

total population = 1E7

成長の制約を取り除くと次に示すグラフの様になります[4]。



これら2つのモデルは、1989年以降、劇的に分かれます。純粋な指数関数的成長においては、普及プロセスは0に張り付く様な平坦な直線の様になるのですが、1988年より前では、2つのモデルの結果は同じ様に見えます。右上のグラフは1960～1990のクローズアップですが、ここでは2つのモデル間に違いはほとんどありません。

このモデルの成長率を計算すると、次の様になります。

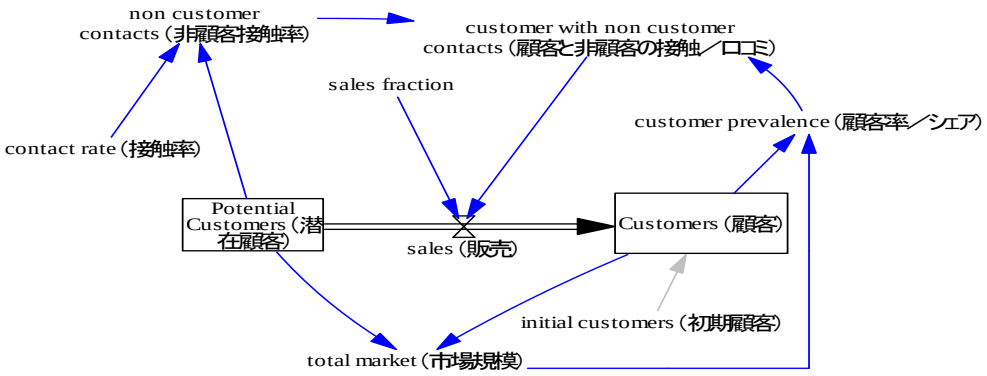
contact rate * adoption fraction = 100 * .005 = .5

即ち50%/yearです。もし、adoption fraction が0.001であれば潜在成長率は10%ということになります。このように、adoption fractionによって、Practitionersのブレイクに大きな差が出てくるのです。

4 細部まで同一のグラフを得るには、積分法としてオイラー法を用いること。

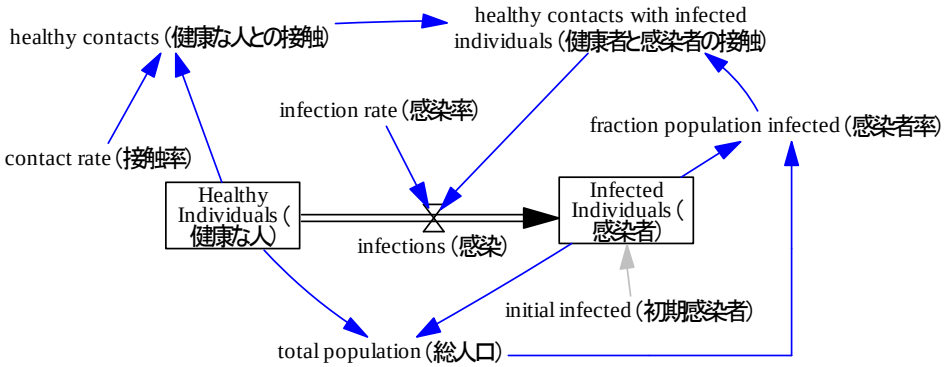
4.4 インフルエンザ・モデル(flu.mdl)

上記は、様々な「普及プロセス」を考えるうえで、共通コアとなるモデルです。ここでは、新製品販売の例を考えて見ましょう。（sales.mdl）



採用は売上と置き換えて考えることができます。そして、採用の振る舞い考えれば、長い間小さい売上が続きます。そして、それらは爆発し、直ぐに崩壊します。「Atari」の様な多くの会社がこのようにしてどこかに行ってしまいました。プロセスはジェネリックであるが、驚くべき成功が暗転するときには、いつも皆（勿論、当の本人たちでさえも）が驚くことになるのです。

もう一つの例として疾病の蔓延を考えてみましょう。（flu.mdl）

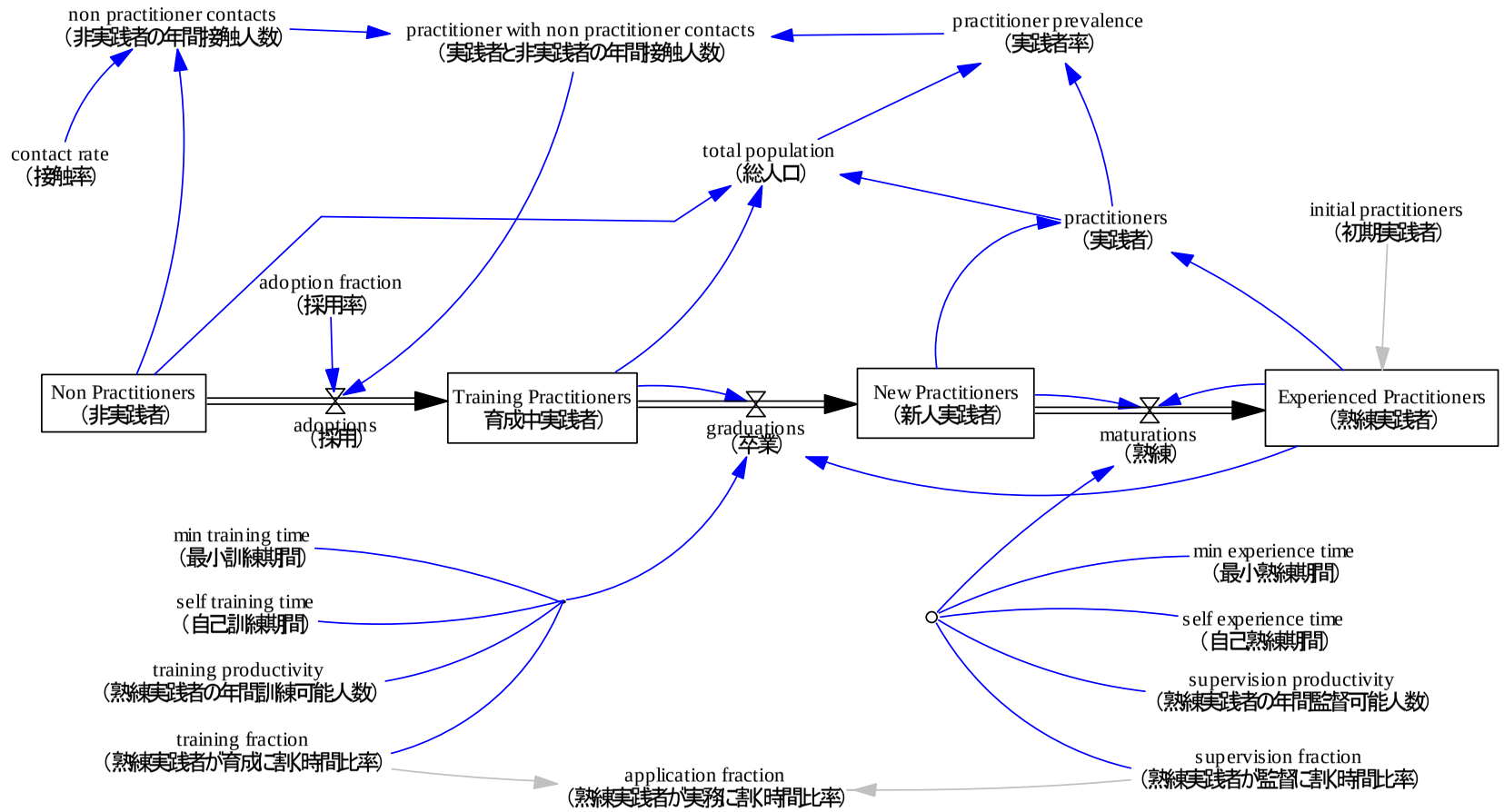


変数名は変わっていますが、これらは両方とも最初に考慮したモデルと全く同じモデルです、また両方とも訴えかけている意味合いも同じです。例えば、これらのどちらかを入念に作るために、症候性、無症候性、伝染性・非伝染性といった、インフルエンザ患者の相違を付け加えようとします。そして、構造を加えるときに強い類似性が残っていることに気づきます。

4.5　普及プロセス(sdgrow2.mdl)

普及率がどれくらい重要かということが分かったと思います。即ち、それが低すぎれば、その技術は普及するまでに多くの時間がかかり、それは他の出来事や他の技術のせいで忘れ去られてしまうでしょう。しかしながら、それをモデル化した様に、採用とは単にツールを拾い上げてそれで仕事をするという問題です。しかしながら、不運にもこれはライフワークにする様なものではありません。そのためには、技術が追求して行くだけの価値がある、と判断した後に、時間を費やして技術を使用できるだけの十分な能力を獲得する努力をすることが必要です。

単に、Non PractitionersおよびPractitionersとする代わりに、Non Practitioners、Training Practitioners、New PractitionersそしてExperienced Practitionersとしてみます。そして、PractitionersはNew PractitionersとExperienced Practitionersの和とするものとします。Experienced Practitionersとは、Training PractitionersからNew Practitionersへの状態遷移を加速するための訓練を施すことができるものとします。(次頁のモデルを参照)


non practitioner contacts
（非実践者の年間接触人数）
practitioner with non practitioner contacts
（実践者と非実践者の年間接触人数）
practitioner prevalence
（実践者率）
contact rate
（接触率）
total population
（総人口）
practitioners
（実践者）
initial practitioners
（初期実践者）
adoption fraction
（採用率）
Non Practitioners
（非実践者）
adoptions
（採用）
Training Practitioners
育成中実践者）
graduations
（卒業）
New Practitioners
（新人実践者）
maturations
（熟練）
Experienced Practitioners
（熟練実践者）
min training time
（最小訓練期間）
self training time
（自己訓練期間）
training productivity
（熟練実践者の年間訓練可能人数）
training fraction
（熟練実践者が育成に割く時間比率）
min experience time
（最小熟練期間）
self experience time
（自己熟練期間）
supervision productivity
（熟練実践者の年間監督可能人数）
supervision fraction
（熟練実践者が監督に割く時間比率）
application fraction
（熟練実践者が実務に割く時間比率）

この図形は少々煩雑ですが実は最初のモデルと同じ基本構造です。人々が育成中実践者から卒業を経て新人実践者となる速度を決定する6つの定数があります。self training timeは、公式訓練を受けない人がPractitionersとして十分に熟練している様になるのに必要な時間です(自習習熟時間)。min training timeは、十分な公式訓練を受けた人が能力を得るのに必要な期間です(最短訓練時間)。Experienced Practitionersがトレーニングのために時間を割くにつれて、training productivityに従って、平均訓練時間はmin training timeからself training timeに変化して行きます。新人実践者から熟練を経て熟練実践者となるための速度についての定式化もまた同様です。

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

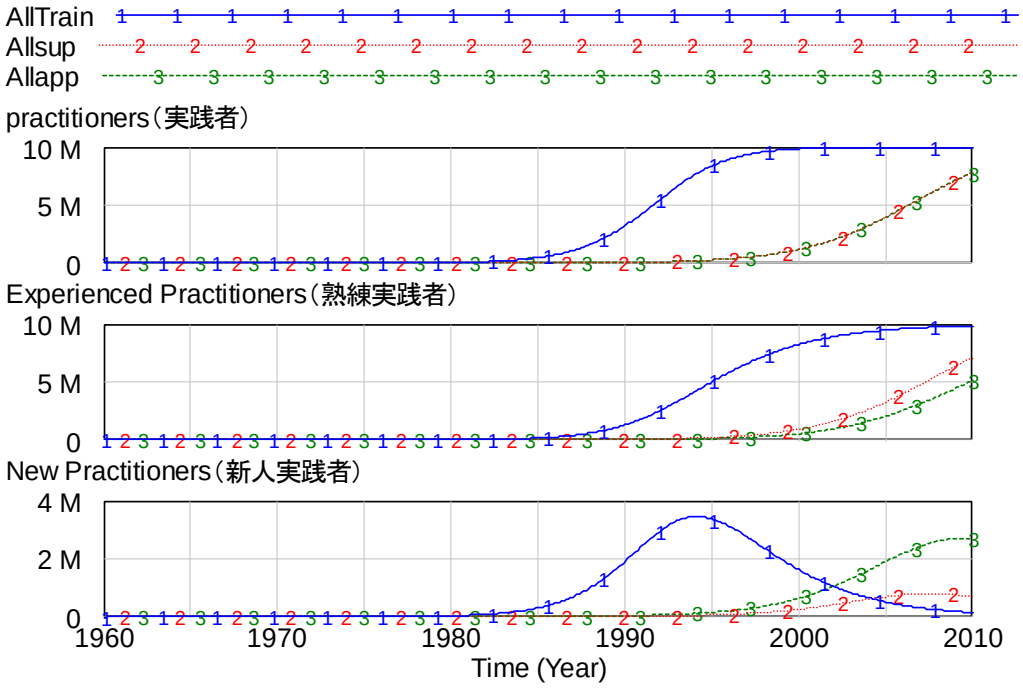
これは、熟練実践者が、この技術の適用(研究を行い、公表し、問題を解決するのを支援すること)に充てる時間の割合を表しています。そして、その時間については、他者を訓練することはできません。これはこのモデルの中で使用されませんが、次の段階で行う精査において考慮します。

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

直接的にトレーニングに専念する人数を増やせばgraduationsのフローを増やすことになります。その速さは、これ以上トレーナを加えてももはや効果がないという時点において、予期される速さと同じ速さまでです。

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
supervision productivityは、1人の経験を積んだ実践者が訓練することができる1年当たりの人々の数です。従って、単位は(Person/Year)Personまたは1/Yearです。
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
このモデルをシミュレーションする場合、次の3つの極端なケースについて行ってみるべきです。
① application fraction=1、すなわち、当該分野での実践に専念し、新しい実践者は自習しなければならないという設定
② training fraction=1、すなわち、新人実践の育成に専念するという設定
③ supervision fraction=1、すなわち、熟練実践者の育成に割くという設定
その結果、次の3つの振る舞いを観察することができます。



監督や適用にすべて注意を割り当てた場合、両方とも成長し飽和するスピードは大変ゆっくりとしたものにしてしまいます。そして、唯一の違いは、Experienced Practitionersの割合が違うということです。もし、Experienced Practitionersがすべての時間をNew Practitionersの育成に充てたなら、Practitionersのうちの大きな割合がExperienced Practitionersとなります。しかし、Experienced Practitionersは何もすることがなく、経験は積んだものの有用な仕事ができない、より経験を積んだ人々にただなるばかりという状況である。

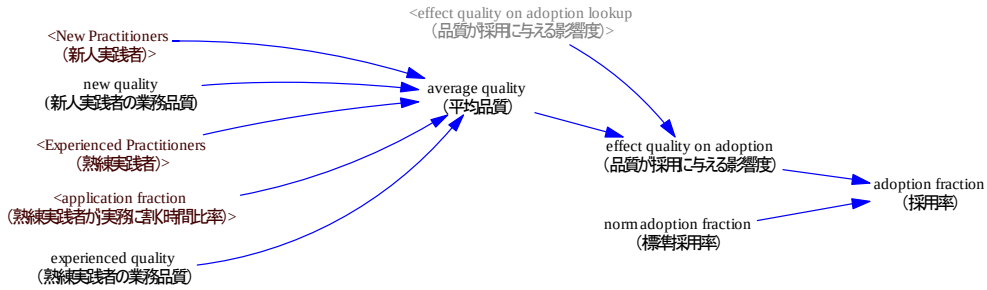
もし経験を積んだ人々が彼らの時間を初心者のトレーニングにすべて費やせば、学問分野の成長のうえで深く重大な結果をもたらします。興味を示す人々は早期に熟練でき、技術を使用し始めることができます。これは面白い結果である一方で、モデルに足りないものがあることを示唆しています。もし、経験を積んだ人が単にトレーニングだけを行っていれば、実践はもっぱら新人実践者によって行われていることになりますが、新人実践者は熟練実践者と同じだけのパフォーマンスではないはずです。

4.6 仕事の質(sdgrow3.mdl)

新技術を採用する人々の意欲は、技術を用いることによって期待されるメリットと、技術を使いこなすための学習の困難さ等によると考えられます。技術の価値の信奉者が多くいることは重要ですが、もし技術が著しく価値のある結果を示すことができなければ、その技術はブレイクしません。ここでは、技術の成功の基準として「仕事の質」という概念を導入し、「仕事の質」を決定するにあたってはNew PractitionersとExperienced Practitionersを区別します。

ここで、「質」とは、成功裡に実現されるプロジェクトの割合を表すものとします。プロジェクトにおいて悪い決定がなされたときは、プロジェクトは開始されたとしても途中で軌道から離れてしまい、実現することなく放棄されてしまいます。そして、仕事の質がadoption fractionに影響を与えるものとします。

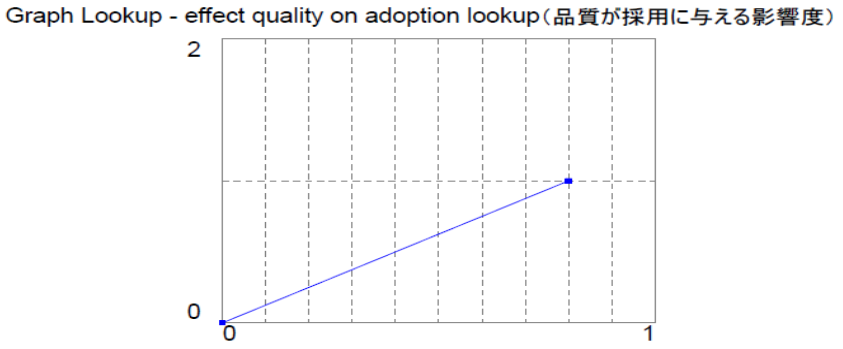
ここでは、average qualityとそれによる採用率に対する効果を得るために新しい変数を加えます。



次の方程式を加えます。

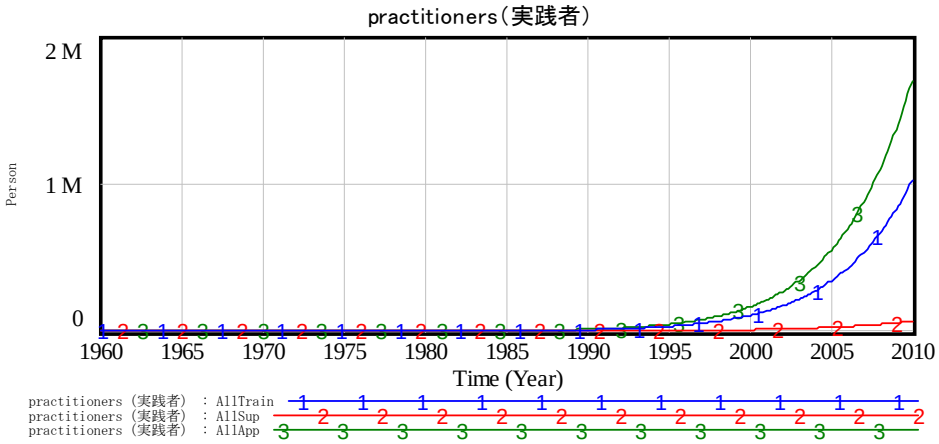
```
adoption fraction = norm adoption fraction * effect quality on adoption
Units: Dmnl
average quality =XIDZ( New Practitioners * new quality +
 Experienced Practitioners * application fraction *
 experienced quality, New Practitioners +
 Experienced Practitioners * application fraction,
 experienced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

XIDZ機能は、適用分数が0である場合に、特別事件に経験を積んだ質にaverage qualityをセットすることによって、計算エラー(オーバーフロー)を防ぎます。



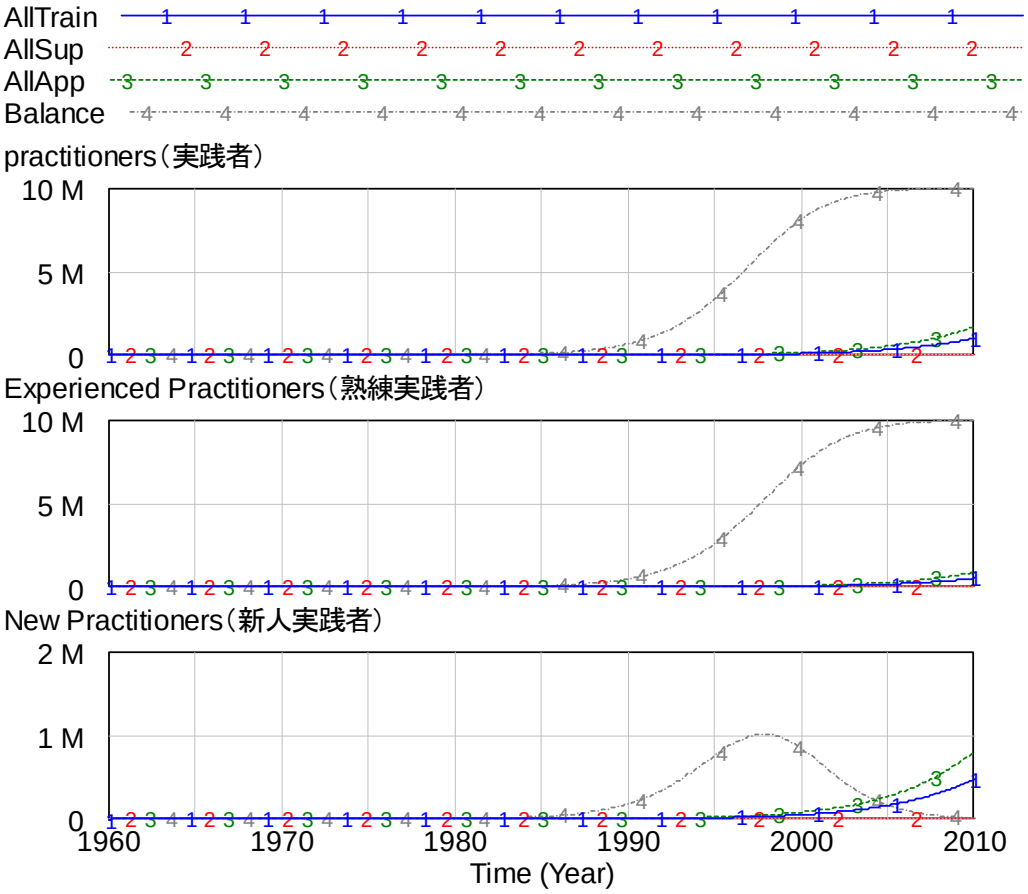
effect quality on adoption lookup((0,0),(0.8,1))
effect quality on adoption = effect quality on adoption lookup(average quality)
Units: Dmnl
experienced quality = 0.8
Units: Dmnl
new quality = 0.4
Units: Dmnl
norm adoption fraction = 0.005
Units: Dmnl

上記の3つの極端な例をこのモデルを使ってシミュレートした結果は次のとおりです。



Practitionersにすべての時間を適用に費やさせることは、いまのところ最良の成長戦略です。誰もが、実践の中で学びます。しかし、成長率はすべて最新のモデルのものに比べて遅いことが分かります。理由は単純です。新人実践者だけが適用を行っている場合、仕事の質は低くなります。その結果、新しく関心を抱く人は少なくなります。このモデル中の成長を最大化するために、適用と教えることの間のバランスを得ることが必要です。

Supervision fractionとtraining fractionをそれぞれ0.1に設定すれば、より良い結果を得ることが分かります。

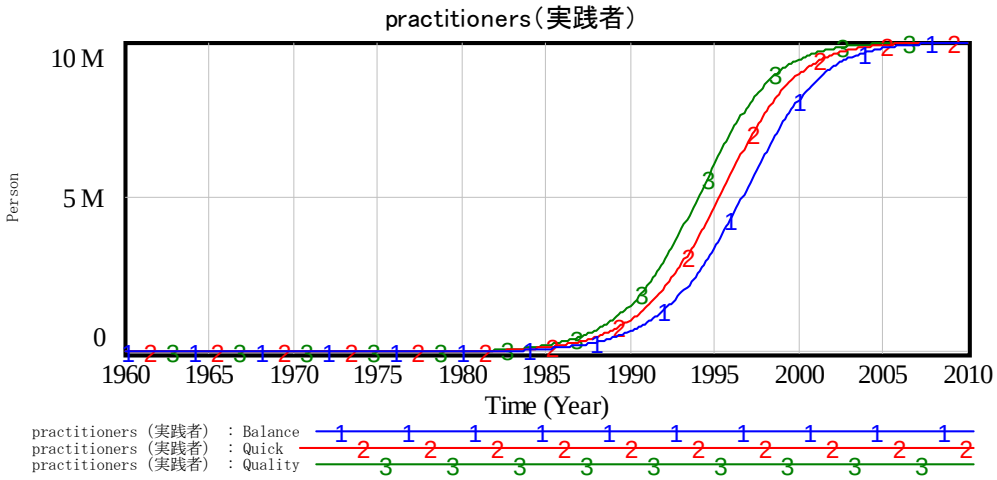


ここでのポイントは、現実性を持たせるために構造を付け加えたのと同じ様に、人々の育成ということに関して、狂気の滝のごとき単純で行過ぎた戦略を避けることにあります。

4.7 ソフトウェアツール

このモデルによって、新しいソフトウェアツールの役割を考えるためのフレームワークを見出すことができます。ソフトウェアツールが提供するものは2つあります。即ち、使い易さと理解の深さです。1つ目は熟練するために必要なトレーニング時間を減らすことです。2つ目は、新人でも経験を積んだ者でも、モデラーの仕事の質を高めることができます。

経験を積んだモデラー10%の時間をトレーニングと監督の両方のために費やし続けることは、自己訓練時間と最小訓練時間を切り詰める努力となります。(これを"Quick"というRunネームとします。)それから、もう1つの実験のための変化を加えるために、new qualityを0.6にセットします。(これを"Quality"というRunネームとします。)



使いやすさと質は両方とも普及速度に重要な影響を及ぼします。Vensimは、使いやすさおよびより高品質の結果の両方を提供することを目指しました。また、それがシステムダイナミックスの成長を促進することを望みます。

4.8 結論

この章では、多くの仮説を書き出すことからスタートしました。本来は統一化されたフレームワーク中で扱われるべきこれらの仮説を調査するために役立つモデルを開発しました。この方法の長所は、スタート時点から打つべき「施策」に注意を集中することができるという点です。そして、異なる人々の間で問題の考え方がずれない様に、モデルが考えの連続性を保つことに役立ちます。これはとても有用です。結果を実現してゆくために、最も大きな問題となることの1つは、利害関係者に対して政策の結果を言葉で説明する際に起こります。仮説に対して絶えずモデルを関連づけることによって、議論の中でアイディアの共有の問題は終始議論の中で取り扱われているので、後になってアイディアの共有ができないことが原因で、施策が挫折してしまう様なことは起こり得ません。

ここで、このアプローチの注意事項について整理してみましょう。今までシステムダイナミックスの学問分野の発展に関する問題考えるために、単純な概念のモデルを構築してきました。開発したモデルから洞察を得ることができましたし、仮説のうちのいくつかを調査することもできました。しかしながら、それらから結論を導き出すことはできません。何故ならば、単純なモデルから洞察を得たとき、一度立ち止まり、「起こっていることはこれなのか？」ということを自問自答する必要があります。ある場合には、答えは「イエス」かもしれません。また、モデルは、現実を反映していて、その理解のための新しい根拠を与えることもあるでしょう。その場合答えは恐らくあります。ここで見てきたダイナミックスは妥当性のあるものでしたが、実際に起こっていることを本当に表しているかどうかについて確信はほとんどありません。妥当性がある様には見えるが、全く間違っているモデルで終わってしまわない様に、更に話を進めて、データの活用と実現性の点検を行う必要があります。

第５章
自社の生産能力（キャパ）を考慮したセールス拡大モデル

5.1 はじめに

4章では、変数名を変更するだけで、システムダイナミックスという学問分野の発展モデルを新製品の投入に応用する方法を説明しました。本章では、新製品の投入に関して、会社と市場を関係づけるとともに生産のダイナミックスを付け加えます。そして、市場の需要に対応する戦略を明らかにして行きます。更に、6章では、2社以上の競争を取り扱うためにこのモデルを更に拡張します。

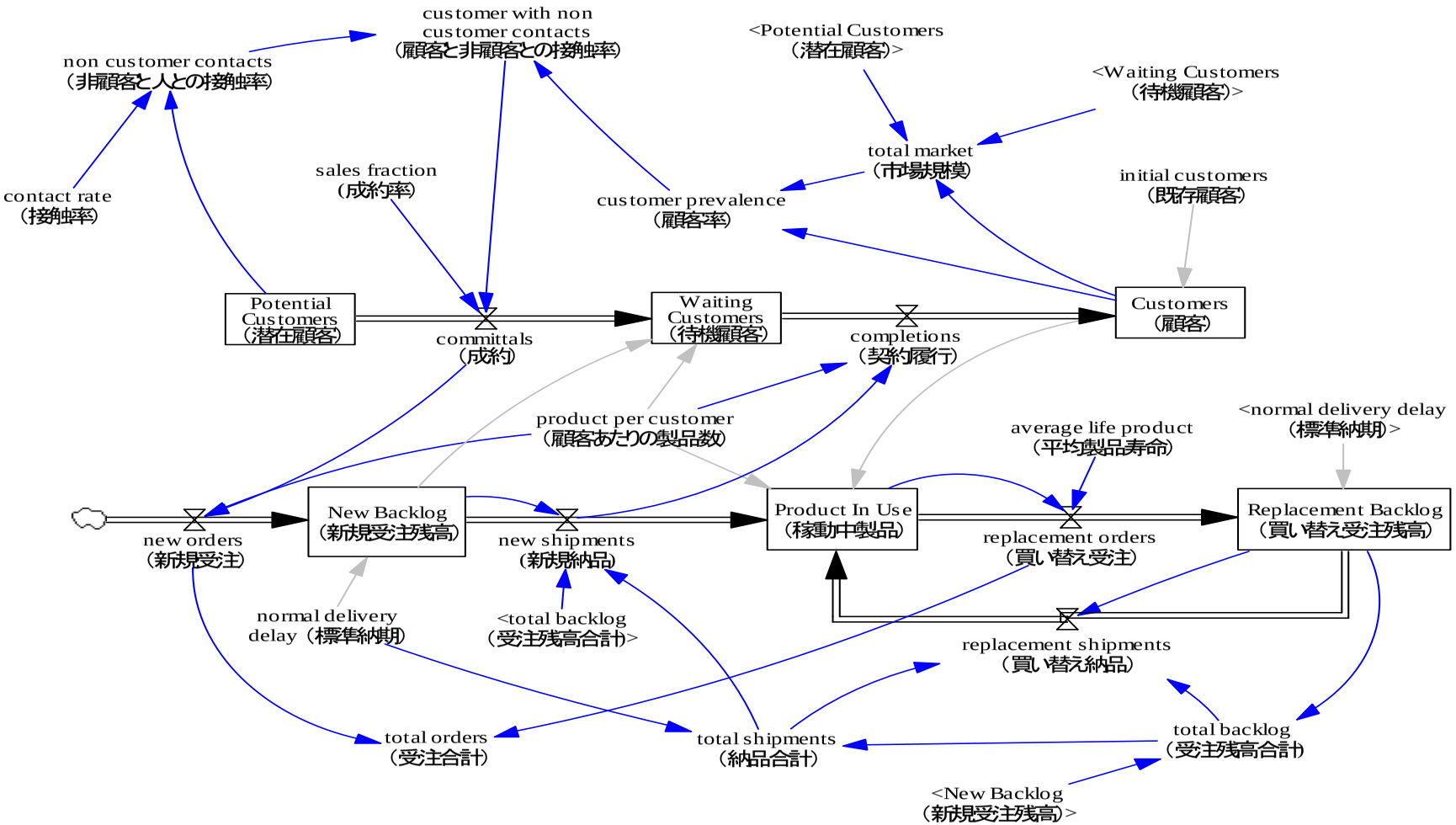
5.2 販売および買い替え(prod1.mdl)

4章の流行伝播モデルを利用するために、現実性のある新製品購入のモデルを取り扱うためには、モデルを更に改善する必要があります。4章のモデルにおいて、salesはその製品が販売された人数を表しています。しかしながら、ビジネスとして考えるならば、製品を採用した人数と実際に購入された製品の数を識別して考える必要があります。製品を採用した人数と実際に使われている製品数という2つの概念は類似していますが、それらを分離したうえで、プロセスに従って単位を整理したうえで眺めることは有用です。

製品を適用する人と製品そのものを区別することで、例えば、受注残や買い替えによる購入の様に、現実のシステムにおいて重要な特性を表わせる様になります。一旦誰かが購入を決定しても、必ずしも直ちに入手できるとは限りません。新しい市場が急激に立ち上がっている場合では、デリバリの遅れは一般的に起こるものです。デリバリを待っている人々は製品をまだ入手できていないので、恐らくその人たちは商品について、他人に商品の評価について話をしたりはしないでしょう。もう一つ、置き換えによる需要を見落としてはなりません。人々は製品を購入したらそれを使用し、使い古してしまうか、旧式になってしまった場合には、その製品を置き換えようとします。ビジネスの継続性を考えればこれはとても重要なことになりますし、産業が長期間繁栄するための条件としてこのことは見落とされるべきではありません。

これらの問題を扱うために、2つのストックとフローが並列する構造を設定します。1つは顧客を表すストックであり、もう1つは製品を表すストックです。潜在的な顧客が初めて製品を注文する場合、それをnew orderと呼びます。そのオーダーはNew Backlogに入り、normal delivery delayの時間遅れで納品されProduct In Useになります。一方、製品の出荷を待っている間は、顧客はWaiting Customersにいます。new shipmentsがなされた場合にCustomersとなり、始めてその顧客は製品に関してのあれやこれやアクティブに話をする存在となります。

ここでTIME STEPを.0625として5年間モデルを実行してみます。


customer with non customer contacts
(顧客と非顧客との接触率)
non customer contacts
(非顧客と人との接触率)
<Potential Customers
(潜在顧客)>
<Waiting Customers
(待機顧客)>
total market
(市場規模)
contact rate
(接触率)
sales fraction
(成約率)
customer prevalence
(顧客率)
initial customers
(既存顧客)
Potential Customers
(潜在顧客)
committals
(成約)
Waiting Customers
(待機顧客)
completions
(契約履行)
Customers
(顧客)
product per customer
(顧客あたりの製品数)
average life product
(平均製品寿命)
<normal delivery delay
(標準納期)>
new orders
(新規受注)
New Backlog
(新規受注残高)
new shipments
(新規納品)
Product In Use
(稼動中製品)
replacement orders
(買い替え受注)
Replacement Backlog
(買い替え受注残高)
normal delivery delay (標準納期)
<total backlog
(受注残高合計)>
replacement shipments
(買い替え納品)
total orders
(受注合計)
total shipments
(納品合計)
total backlog
(受注残高合計)
<New Backlog
(新規受注残高)>

ここで、少々構造を加えましたが、流行伝播モデルと同じ動的な特徴がまだ残っています。その中心となるフィードバックループは不変であり、そのループがシステムの成長や飽和(より多くの顧客、より多くの接触、より多くの販売、そして誰をも顧客にしてしまうこと)をドライブします。そして、潜在的なバッファーである待機顧客(Waiting Customers)があります。待機顧客は活動的ではありません。すなわち、商品についてあれやこれや話をすることや新しく製品を購入することがありません。そしてそれが成長を阻害することになります。しかしながら、最初のシミュレーションでは配達遅れ時間を短く設定しているので、このことは重要なインパクトを持たないでしょう。しかし、生産モデルを導入すると大変重要になってきます。

方程式は次のとおりです。

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

システムダイナミックスの様な複雑な方法論について説明することに比べれば、誰かに製品を見せて素晴らしさを理解させるのは、はるかに簡単だと考えられるのでcontact rateは4章のオリジナルの流行伝播モデルに比べて大きな数値としました。

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

ここでは、顧客をかなり大きな値から始めています。もし、もっと少数の顧客数から始めようとすれば、普及のプロセスを合理的に短縮するための広告等の活動が必要になります。(これは自習用の演習として自分で確かめてみてください。)

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とreplacement ordersは両方とも同じ製品の供給源に対して競合する関係にあります。(すなわち、利用可能なキャパによって制限されるということです。)それらの製品がそれまでに要求された量に比例して配分されるということで定式化します。これは、稀少資源の分配を表すための単純で一般に有用な方法です。定式化、

new shipment=New Backlog/normal delivery delayが同じ結果を与えます。このように定式化することによって、生産セクターをよりシンプルに追加できる様になります。

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
```

ACTIVE INITIAL関数はWaiting Customersを含むために、シミュレーションの実行エラーを避けるために必要です。この機能は、シミュレーションでは第1項の式を評価しますが、初期化においてストック変数に初期値をセットするときは第2項の式を評価します。Waiting Customers、Customersは、シミュレーションの開始時点では値は小さいので、この方法は初期値の相互依存を解消するための合理的な近似です。

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

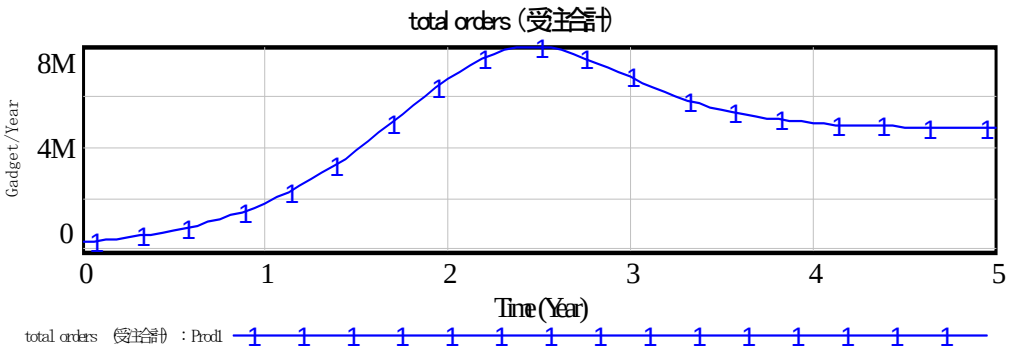
このモデルは少々上手く作ってあるので、方程式total shipmentを修正するだけで生産構造を付け加えることができます。このように構造を完全に分離することは、教育的な目的としては望ましいことかもしれませんが、目標としてあまり目指すべきものではありません。フィードバックが綺麗に分離できず、お互いのセクターに浸透しているのが普通です。また、異なるセクターを過度に切り離そうとすることで、非現実的なモデルになってしまう場合もあります。

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

Waiting Customersは受注フローと納品フローのバランスの中で初期化されます。このモデルをシミュレートする場合、あなたのオプション・セッティング(シミュレーションの制御の実行用コメントに"-c"を指定)によっては、次の様な警告が出力されます。

これらの警告はACTIVE INITIAL関数を使っていることに関係があります。total marketの値はモデル中のストックを初期化するために使用され、値は1E7でした。一旦、すべてのストックが初期化されたならば、total marketはACTIVE INITIAL関数のアクティブな項である1.013E7で計算されます。追加された変数については明示的にはACTIVE INITIAL機能は使用されていませんが、2通りの計算が進められ、違った値がレポートされます。これらの値の差は小さいので懸念される様なことの原因になる様なことはありません。

この基本型は、前の章において流行伝播モデルで議論したのと同じ振る舞いを生成します。しかしながら、replacement ordersを追加したために、total ordersは新しい違った側面を持つ様になります。



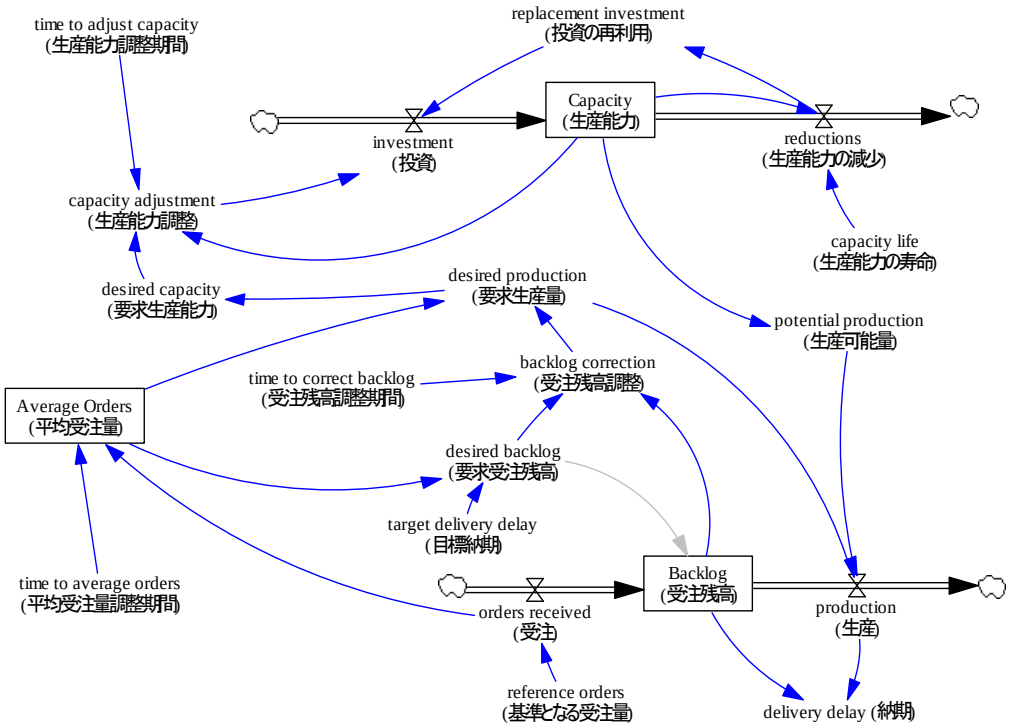
ここで、total ordersは急速に成長します。そして0に落ちるのではなく、ちょうど500万弱（装置/年）の値を保持する様になります。これらは顧客が使い続けている限り発生する買い替え需要によるものです。Product In Useは、もしすべての人がその製品を持っているとすれば達するであろう1000万には届きません。これは、一旦製品を置き換える決定を下しても、置き換えには時間がかかるからです。常に製品を待っている人々がいるのです。

5.3　生産(prod2.mdl)

生産側をモデル化するためには、キャパ(どれだけの量を生産できるか)を決定し、納品によって注文が履行されてゆくことを見てゆく必要があります。キャパの決定は生産目標に基づく目標の調節として定式化します。これは、労働力-在庫モデルで使用される労働者採用のための定式化に非常に似ています。このモデルにおいてキャパは、資本と労働力の両方を含む複合的な生産能力と考えられます。従って、キャパを調整するための時定数は、設備等の資本的な装置を獲得するリードタイムの長さを反映してより長いものとなります。

注文のモデル化は受注残高(注文は出されたが、まだ履行されていない)によってなされています。受注残高は多くの点で在庫の反対の意味を持ちます。この点で生産は労働力-在庫モデルに似ています。このモデルのもう一つの違いは、目標キャパや受注残高を計算すれば注文の流れは平均化されることです。労働力-在庫モデルでは、販売に基づいて目標生産を直接定式化しました。これは単純な定式化ですが現実的ではありません。普通は調子のよい日あるいは調子のよい月に基づいて、将来の計画をすべて立てたりはしません。注文の短期的な変化は、均されてしまい将来の計画に対して影響はほとんど与えません。

ここで、Vensimの機能を使って、新しいビューを作成し生産セクターを加えます。



このモデルの生産セクターは最初のビューから完全に分離されて、スタンド・アロンのビューとして定式化されています。モデルのうち、このビューの振る舞いを分離してテストすることができる様にしたのです。その後、2つのモデルをリンクし、どの様に動作するかを試します。この手順は開発を見通し良く進めるために有用な方法です。一般に、モデルのサブシステム間の境界はこの例の様に綺麗にはなりません。しかし、構造の一部がどの様に作用するか分離して確かめるためにモデル変数をテスト入力に置き換えることは常に可能です。

このセクターの方程式は次のとおりです。

Average Orders = INTEG((orders received - Average Orders)/time to average orders, orders received)
Units: Gadget/Year
Average OrdersはSMOOTH関数を使用して書くこともできますが、平滑化のプロセスの性質を強調するために明示的に積分を使用しています。
Backlog = INTEG(orders received - production, desired backlog)
Units: Gadget
backlog correction = (Backlog - desired backlog)/ time to correct backlog
Units: Gadget/Year
モデルを均衡状態でスタートさせるために、Backlogはdesired backlogで初期化します。
Capacity = INTEG(investment - reductions, desired capacity)
Units: Gadget/Year
モデルを均衡状態でスタートさせるために、Capacityはdesired capacityで初期化します。
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
orders receivedのSTEP関数は、時刻 1 において注文を2倍にしています。STEP関数によるドライブは比較的容易に理解できる振る舞いをシステムに与えるので、テスト入力にSTEP関数を使用することは有用です。
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
target delayは、消費モデル中でnormal delivery delayとしてセットされたのと同じ値

にセットします。数値的には同じでなくても構わないのですが、概念が2つ並列しているのですから近い値にするべきです、

time to average orders = 0.25

Units: Year

time to adjust capacity = 1

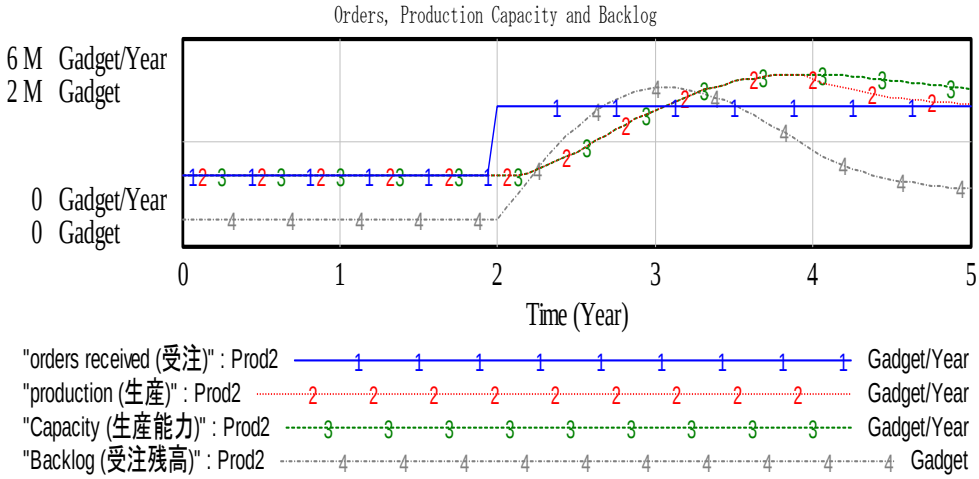
Units: Year

time to correct backlog = 0.5

Units: Year

このモデルをシミュレートした場合の振る舞いは1章の中の労働力-在庫モデルに似ています。

受注残高は受注がステップ的に増加した後一年間増加します。つまり、生産が受注を超えるまでの間、受注残高が増加します。生産が受注を超えると、受注残高は減り始めますが、生産能力はオーバーシュートして増加します。そして、4年目を迎えるまでに、期待生産量が下がると、余剰の設備が発生してしまっています。



実験として、target delivery dalayを1年にセットし、何が起こるか確かめてみます。これは大抵の製品にとっては非現実的な高い値ですが、面白いダイナミックスを生じさせます。

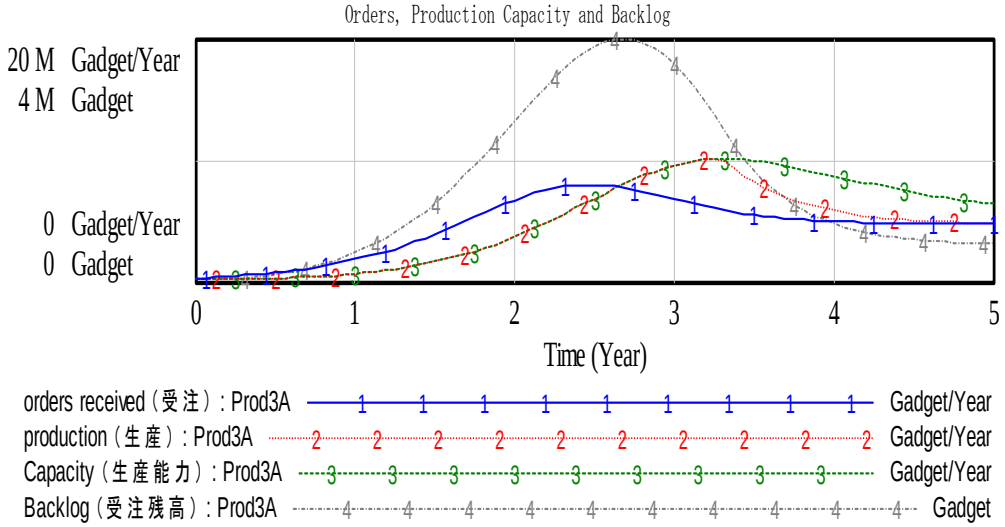
5.4　2つのセクターを組み合わせる（prod3.mdl）

これから行いたいことは、これらの2つのモデルを結合することです。ステップ・バイ・ステップで進めて行くことにします。まず、受注をドライブするために「顧客」が多くなったことによって生み出された受注を使用します。それから、total shipmentsをドライブするためにproductionを使用します。

最初にreference ordersを削除します。次にモデル変数ツールを使用して、ビュー1からtotal ordersとその原因となる変数(new ordersおよびreplacement orders)を加えます。ビュー1に切り替えて、(ポインターでハイライトしながらEdit>Cut、あるいはCtrl-Xを押下します)からのtotal ordersをカットします。ビュー2に戻りorders receivedにtotal ordersを接続し、orders receivedの方程式を入れ替えます:

orders received = total orders

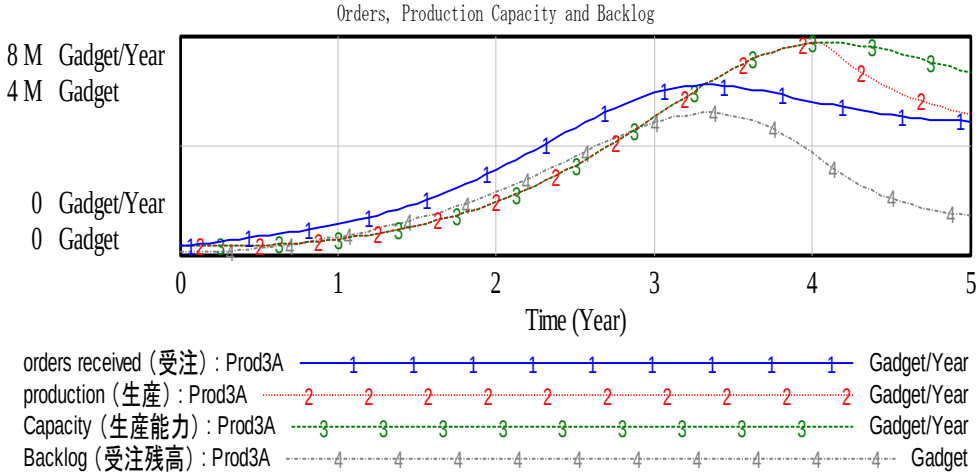
さて、モデルをシミュレートしてみましょう。受注のテスト入力(reference orders)の代わりに、入力として最初のモデルからの出力を使用しています。(Vensimに付属するモデルprod3.mdlを用いている場合、下に示す様な結果を示さないということに注意してください。必要とする結果を得るために、上述した様にprod3.mdlのモデルにおいてtotal shipments=total backlog/normal delivery delayを修正する必要があります。あるいは、prod2.mdlのモデルに対して上述の作業を行う必要があります。結局、次の様な結果を得ることができます。



受注残高は2.5年間増加します。そして、新規の受注が低下するにつれて減少し始めます。(2年目の受注のピークに遅れて)3年目に生産のピークが現れ、その後、余剰設備が発生します。ここではまだ、消費側でtotal shipmentsはtotal backlogをnormal delivery delayで割った値に等しいという仮定を置いています。これは、普及のプロセスが生産設備の制限に影響されないことを意味します。2つのサブモデルを連結するために、次の様に変更します:

total shipments = production

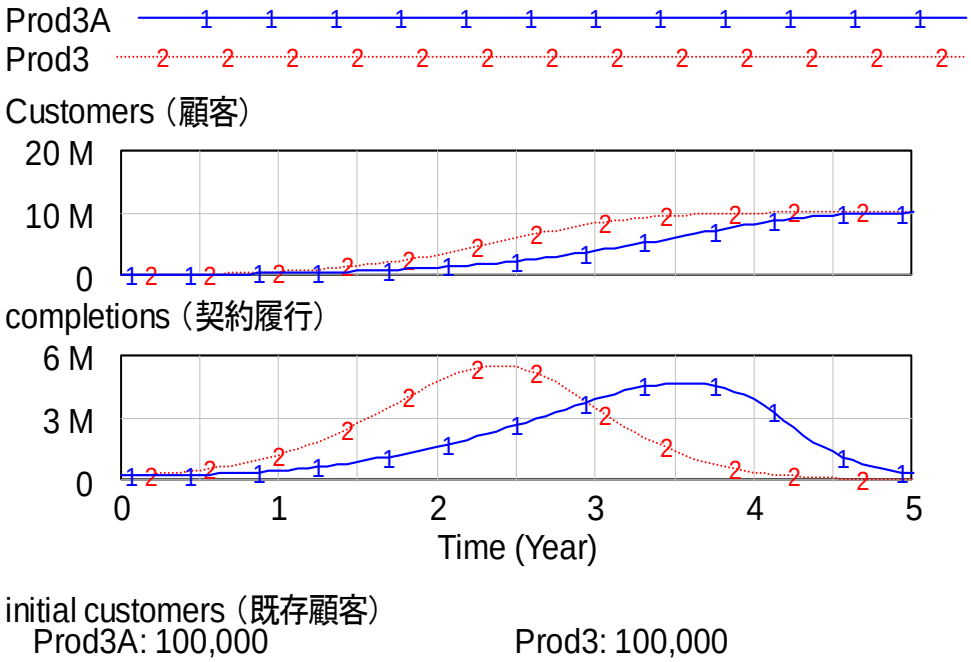
そして図形を適切に整えてください。新しいモデル(実行の際には、新しいRunネームを必ず使用すること)をシミュレーションすると次の様になります。



「生産」と「受注」のピークは、遅れが生じていますが、以前ほど極端ではありません。受注残高のピークは300万未満です。前の実行では大体400万でした。普及のプロセスが、キャパ増強のレートで制限されているのでこの様になるのです。

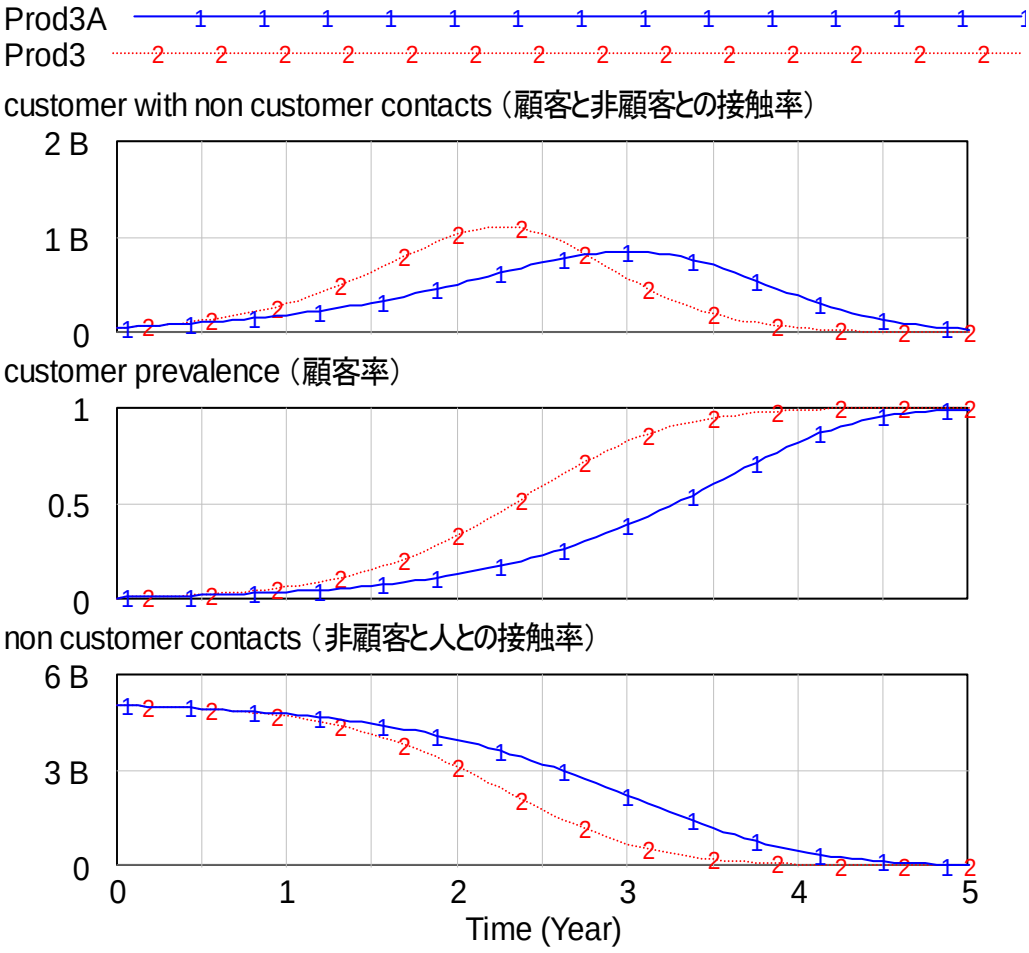
5.5 Ｒｕｎ結果を比較する

ここで、最後の2つのRunの違いを調べておくことは役に立ちます。2つのRunをロードしたまま、Customersを選択して、メニューの「直接原因グラフ」をクリックします。



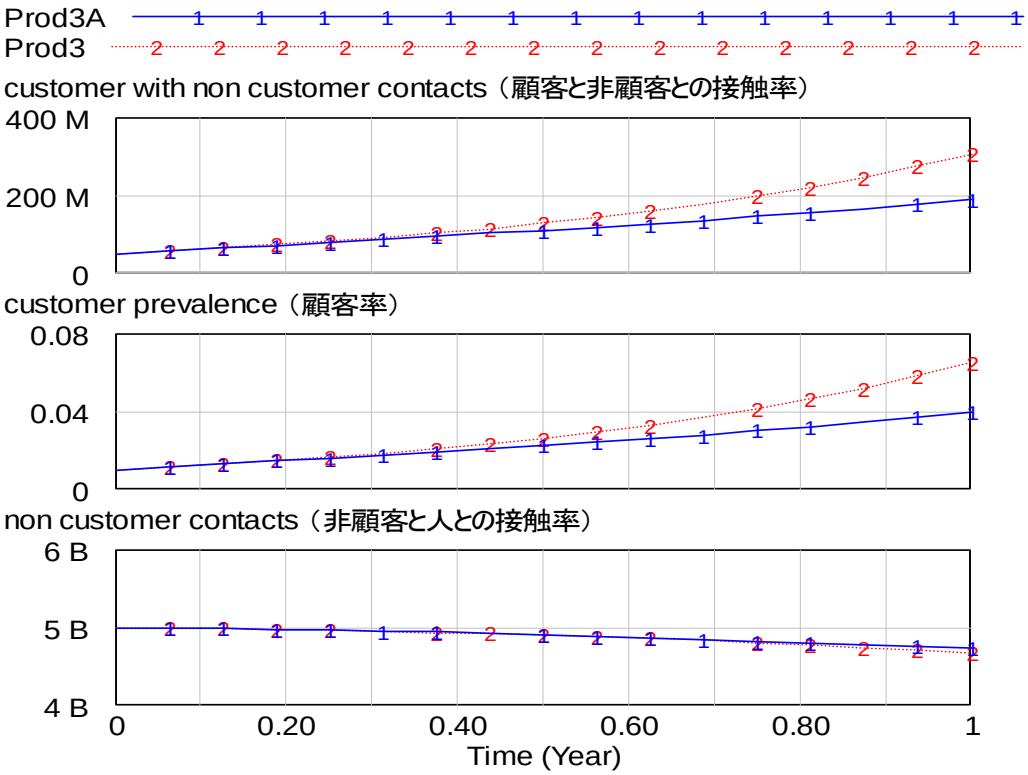
initial customers (既存顧客)
Prod3A: 100,000　　Prod3: 100,000

次に、completionsをダブルクリックして、直接原因グラフをもう一度クリックします。順次このように調べて行くと有効なパスは、Customers、completions、new shipments、new orders、committals、customer with non customer contactsとなります。最後の部分のグラフは次の様になります。

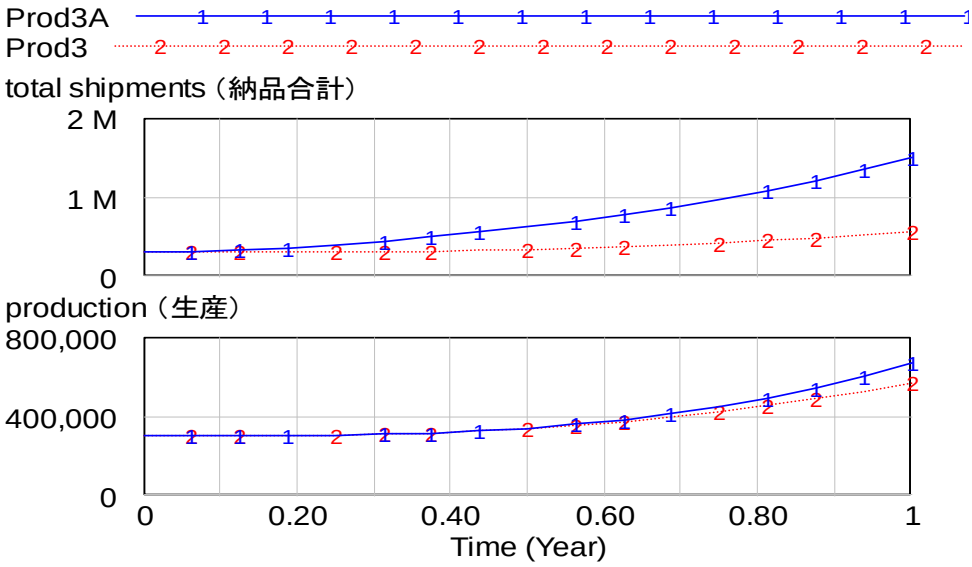


顧客率は、非常に長い間低い値に留まり続けます。普及率が低いので、非顧客との接触は非常に限られたものとなっています。そして、その製品の評判や誰かがその製品の購入を決めたという話を聞くこともほとんどありません。

何が顧客普及の原因になっていて、何がそれを解決する引き金になるのかということを理解するために1年目にフォーカスしてみます。時間軸を0年から1年に限定して表示してみましょう。(方法はチュートリアルの7章を参照)。再び直接原因グラフをもう一度クリックしてみると、より幅の狭い時間レンジのアウトプットが表示されます:



狭い時間範囲を選択したまま、customer prevalenceに影響を与える変数をトレースしてみよう。有用なパスはcustomer prevalence, customers, completions, new shipments, total shipmentsです。最後の直接原因グラフは次の様になります。



これは以前のモデルにはなかった新しい結び付けです。納品合計は生産に依存しています。そして、受注が増加していることを知覚することやキャパを獲得することの遅れにより、生産のスピードを上るためには時間がかかります。

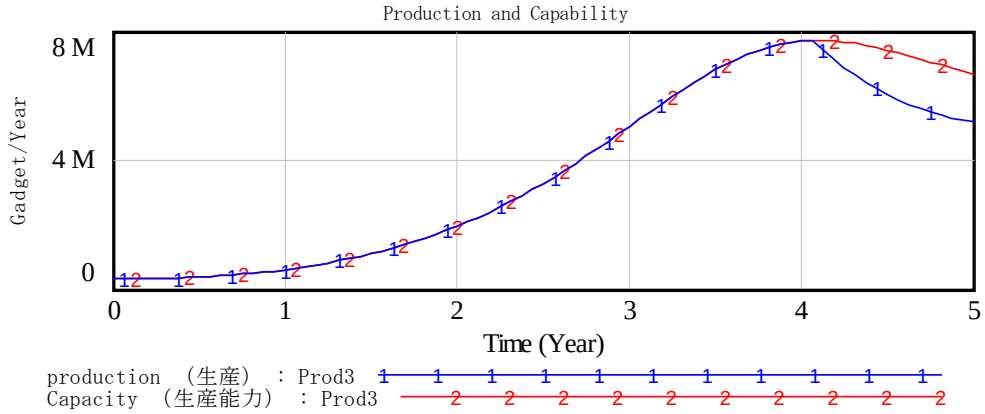
生産が間に合わないということは、製品を手にできる人は少ないということ意味します。そして、これは結局のところ需要を作りあげるのにより長い時間がかかるということを意

味します。ある意味では、これは良いことです。すなわち、マーケットが飽和するにつれて発生するキャパのオーバーシュートを軽減するからです。しかしながら、競争という見地からは、このことは問題になります。これは6章の中で説明します。

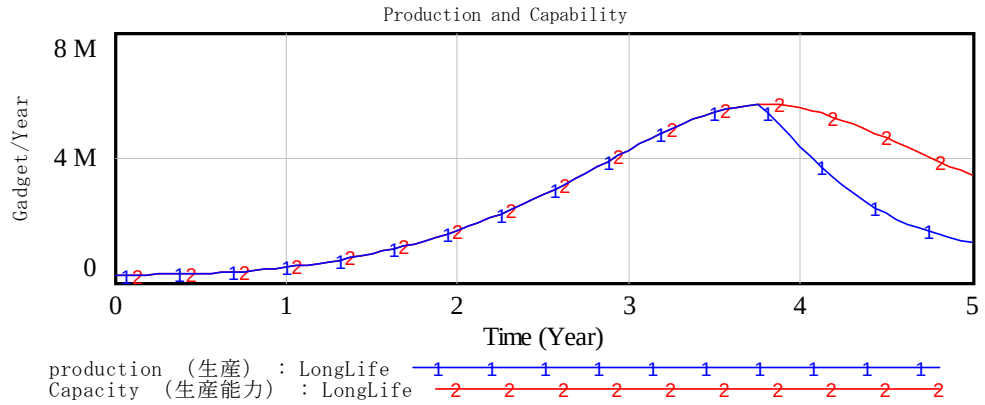
5.5.1 実験

このモデルを使ってできる実験はいろいろあります。モデル中のパラメータを変更してみて、システムの振る舞いの変化を見てください。例えば、より積極的にあなたの受注残高(backlog)を管理した場合の結果を見るために、time to correct backlogを減少させてみるかもしれません。あるいは、キャパを調節する速さをより早くしたり、より遅くしたりしてみて結果を見るためには、time to adjust capacityを変化させれば良いでしょう。

average product lifeを大変大きな値に変えてみることは大変面白い実験になります。基本Runのpruductionとcapacityのプロットをとってみると次の様になります。



次に、基本的に製品が消耗しないとしたときのプロットを見てみましょう。



するとはるかに大きな余剰キャパが発生してきます。製品の寿命を長くしたら、余剰キャパが増えることを見出すことができました。反対に、製品の寿命を短くすれば、過剰キャパがなくなるポイントを見出すこともできます。

まさに瓢箪から駒です。このことは単純なことですが新規市場の製品をマネジメントしようとする多くの人々が見過ごしていることでもあるのです。

第６章

競争のダイナミックス

6.1 はじめに

本章中のモデルは、Vensimの下添え字機能を使用します。下添え字はVensim ProfessionalおよびDSSでのみ使用できる機能です。

5章では、小さな市場成長モデルを構築しました。そして、市場の需要に過敏に対応しすぎない様にして、余剰キャパを持たない様にすることで、よりスムーズに生産セクターが対応できることを見出すことができました。そして、商品の普及による需要から、より長く持続する置き換えによる需要へ移行してゆくことが重要であることも示しました。しかし、競争の観点を導入すると結論は全く異なってきます。競争がある場合、市場への反応が遅いことは競争相手にビジネスで負けてしまうことを意味するからです。

本章では、2社以上のメーカーが存在する場合を検討できる市場成長モデルを作成します。そのために、Vensim ProfessionalまたはDSSでのみ利用できる下添え字機能を用います。本章はPLE版のVensimユーザーにとっては読み物としては面白いかもしれませんが、実際にモデルを構築したりシミュレーションしたりすることはできません。試してみたい場合は、生産セクター・モデルを2つ構築し式を修正することで2つの生産セクターを顧客のビューにリンクすることにより可能と思われますが、そのための具体的方法はここではこれ以上は説明しません。

6.2 モデル(prod4.mdl)に下添え字を付加する

この章におけるモデル作成は、5章で作成したモデル(prod3.mdl)からスタートします。そして、prod3.mdlに対して、ここで説明する様な下添え字を加える作業をするか、既に準備された完成モデル(prod4.mdl)が入っているファイルをオープンしてください。下添え字機能については、Vensimユーザ・ガイドの17章を参照してください。使用する機能のアウトラインだけをここでは説明します。

まず、モデルに下添え字範囲Producerを追加します。そのために、ツールバー上で右端にある「下添え字(ベクトル)」のアイコンをクリックしてください。すると、下添え字範囲の名前を入力することを求められます。PRODUCERと入力して、OKをクリックしてください。すると、方程式エディタが開きますので、"US,THEM"と入力して、OKをクリックしてください。

producer : US, THEM

すると方程式エディタは閉じます。下添え字を扱うために、PRODUCERの下添え字範囲を作成した後、方程式エディタを一旦閉じる必要があります。それを閉じないで、ChooseやNextをクリックした場合、下添え字機能は利用可能になりませんので注意が必要です。

さて、2番目のビューに行って、total orders、replacement ordersおよびnew orders以外のすべてを選択してください。そのためには、total ordersの上にカーソルを置いてシフトボタンを押下しながらクリックし、「編集>一部選択>代行変数以外」を使用すると良いでしょう。

さて、メニューコマンドの編集>下添え字の設定を使います。**Modify Subscripts for -20 variables**というラベルがついた対話ボックスが現れます。PRODUCERを選択して、**Subscript 1** をドロップダウンしてOKをクリックします。添え字PRODUCERは、選択した20変数全部にそれぞれ追加されます。そして、その変数が使用されている式も同時に変更されます。このセット・アップでほとんどの変数について対応は完了してしまいます。しかしながら、これから述べる2変数だけは別対応が必要になります。1つはproductionです。もう1つはorders receivedです。

orders receivedは消費セクターのアウトプットであり、消費セクターには添え字はありませんのでこの変数は添え字を付して使用することはできないのです。この式において、受注が2社のメーカーにどの様に分配されるかを明示する必要があります。最も単純な仮定は、両社が注文の半分ずつを受けるという仮定を置くことです。すなわち、次の様に表すことができます。

orders received[producer] = total orders/2

古い方程式のままでも、orders receivedに関してはエラーメッセージが出力されないことに注意する必要があります。しかも単位チェックもパスしていまいます。古い方程式ですと2つのメーカーそれぞれに、total ordersが割り当てられてしまい、実際の受注よりも多くの受注を割り付けてしまうということです。その様な間違いをしない様に、モデルに下添え字を加える場合は、注意を払う必要があります。

同じ様に注意すべき点がもう一つあります。total shipmentはproductionが代入され同じ値になります。しかしながら、次の様な式は有効ではありませんし、Vensimもエラーを出します。

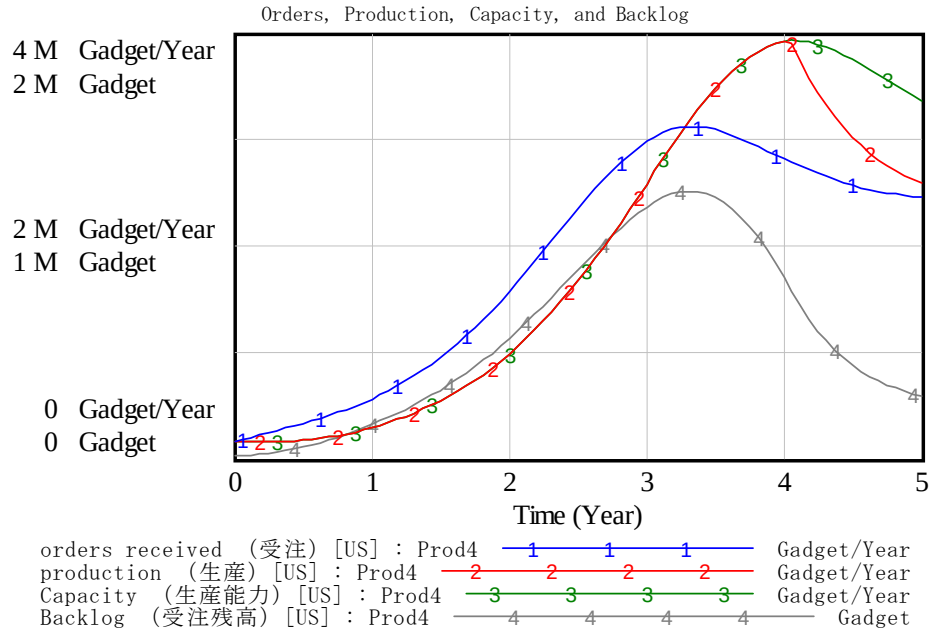
total shipments = production[producer]

なぜなら、2社以上の会社が生産しているので、次の様な方程式にする必要があります。

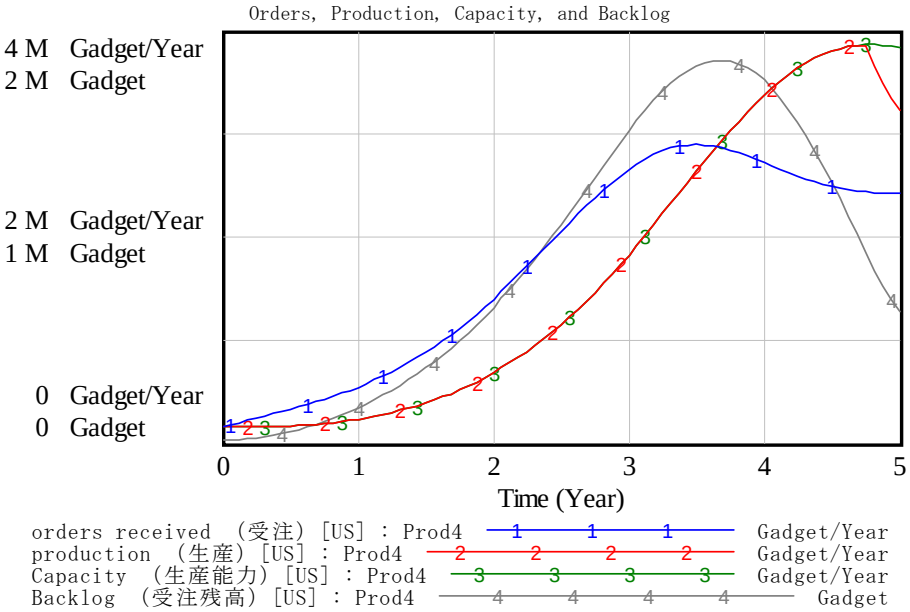
total shipments = SUM(production[producer!])

この方程式は、total shipmentは、PRODUCER各々のproductionの合計であるのことを示しています。感嘆符 ! が添え字に付されると、添え字が取り得る範囲すべてについて合計

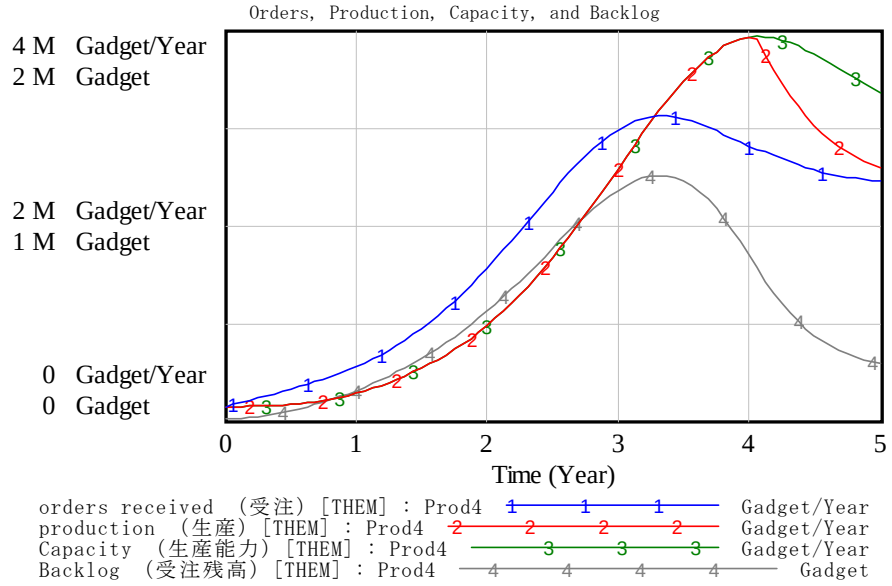
することを意味しています。
　このモデルをシミュレートすると、生産側のモデルで次の様な結果が得られます。



　これは、以前に見た結果と同じですが、スケールが異なっています。当たり前のことで驚く様なことではありませんが、4章における生産セクターの50%を生産している2つの生産セクターがあるということです。もし、2つの生産セクターの振る舞いが同一でない場合は、エラーがないかモデルを調べてみる必要があります。
　ここで、2つの生産セクターのうちのどちらか1つで、キャパを保守的に(慎重にゆっくりと）調整すると仮定してみましょう。生産セクターの1つである(US)は、キャパの調整をより慎重にゆっくりと行うとしましょう。このために、time to adjust capacity[US]を1%から2にしてみましょう。USの生産セクターだけで、この変更行えば、キャパのオーバーシュートがより小さくなり、よりスムーズにキャパを変化させることができることが見て取れます。次の様な結果が得られます。



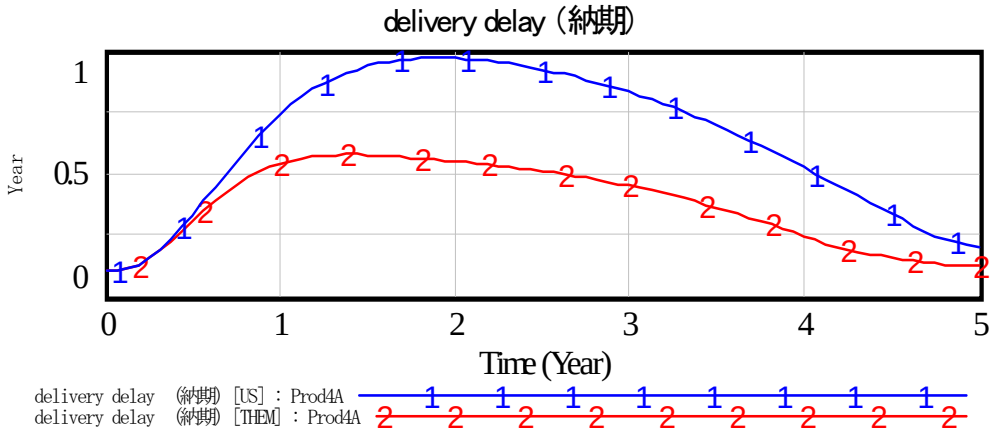
市場における成長のプロセスは遅くなりますが、最終的にキャパが到達するレベルにはほとんど違いはありません。下添え字コントロール・ボタンを使って、PRODUCERのタブをクリックして、USを外してTHEMをオンすることで、同じグラフを使ってTHEMの振る舞いを見ることができます。ここで「自作グラフ」を起動すると次の様なグラフを得ます。



最初のRunと、動き全体のタイミングはほぼ同じです。しかし、余剰キャパは少なくなります。これは、添え字機能によって複数のセクターを持つモデルの集合体において、セクター毎にポリシーを変えたときの効果の観察の仕方を説明している例です。しかしながら、このモデルの振る舞いは明らかに実際的ではないので、この結果にはこれ以上触れない様にします。

6.3 需要と納期(prod5.mdl)

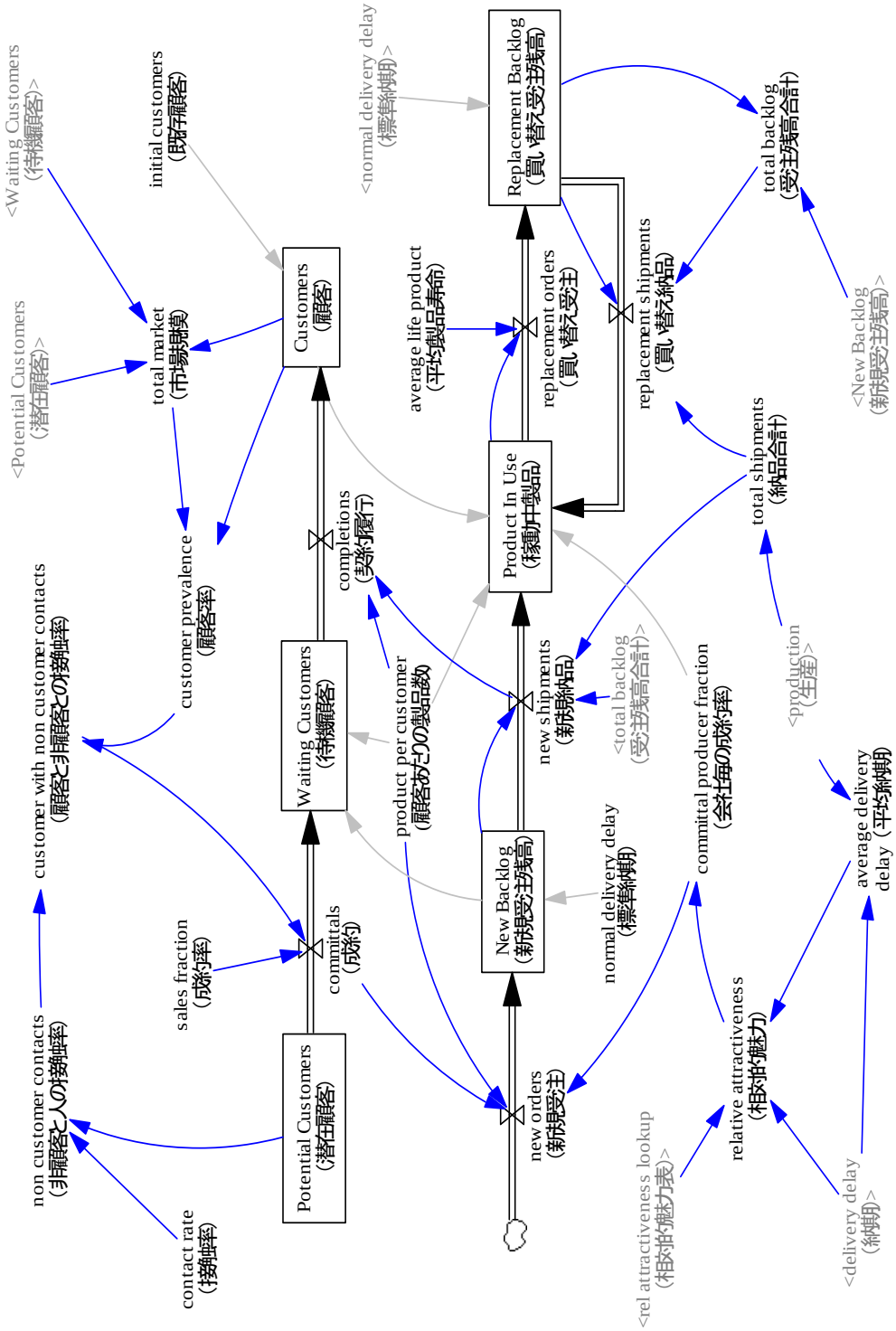
上記の結果では、メーカー2社は両方とも同数の注文を受けています。そして、一方は多くの受注残高を抱え、もう一つの受注残高はそれほどではありません。このとき、2社の納期を調べるためには、添え字コントロールのPRODUCERタブのUSとTEHMの両方をクリックしてオンにすることです。(あるいはAllボタンをクリックしても構いません。) そして、delivery delayを選択して、メニューのグラフをクリックします。



delivery delayは、2社では大きく異なります。しかし、両社は同数の受注を続けます。ここで必要なことは、メーカーが製品出荷のために製品を必要なときにどれだけ使える状態にしているかということと、それぞれのメーカーの受注量との関連づけです。

生産者同士が協力しないとき、需要のシフトが起こるとすれば、それは顧客の振る舞いに由来するに違いありません。生産者の活動のうち、顧客から直接観察できるのは納期だけです。顧客が新規購入にあたって、最も短いリードタイムのサプライヤを選好するという様にモデルを変更します。

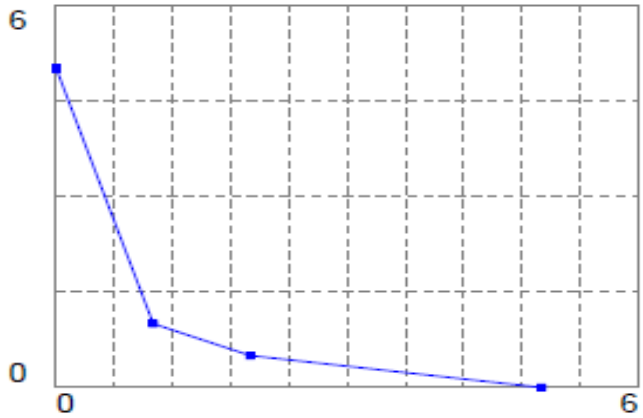
このためにモデルの需要側へ下添え字を部分的に導入する必要があります。この部分について、ユーザーから見た視点と製品から見た視点の2つの並列な流れをモデル化しています。下添え字を、このセクターの受注と納品の部分に適用します。



モデルをより読みやすくするために、初期化部分を仮想に隠す処理をしたことに注意してください。新しい構造は左下の部分に示しています。そして、delay delayから2つの会社のrelative attractivenessが決定されます。その魅力に基づいて、committal producer fractionがそれぞれのメーカーに割り当てられます。下添え字が多くの方程式に加えられたこと以外は、その他の構造は変えていません。消費セクターにおいて今回の変更の影響を受けた式は次のとおりです。

```
average delivery delay = SUM(delivery delay[producer!] * production[producer!]) /
SUM(production[producer!])
Units: Year
```
average delivery delayは、productionによって重み付けさた平均として計算します。
```
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
```
新規需要や置き換え需要のための出荷量の計算は、もともとの定式化と似ているのですが、生産者毎に固有のものとなることが異なります。一度、顧客があるメーカーの製品を使えば、お客様はそれ以降、ずっとそのメーカーの製品を使い続けるものとします。
```
Product In Use[producer] = INTEG(
 new shipments[producer] + replacement shipments[producer] - replacement
orders[producer],
         Customers * committal producer fraction[producer] *
         product per customer)
Units: Gadget
```

Graph Lookup - rel attractiveness lookup（相対的魅力表）




```
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
```

関数ACTIVE INITIALは、同時初期化の問題を防ぐために使用されます。生産セクターの初期化は、注文が既知であることを必要とします。しかし、ACTIVE INITIAL関数を用いない場合、注文の計算は、初期化されていることを必要とします。初期値に関して同時計算を要してしまうことを避けるために、必要な変数に中立の値をセットしてしまうことは有効な政策です。この場合、自然な変数としてセットされるのはrelative attractivenessです。(何故ならば、それは中立的な値である1を持つからです。)

```
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)
```

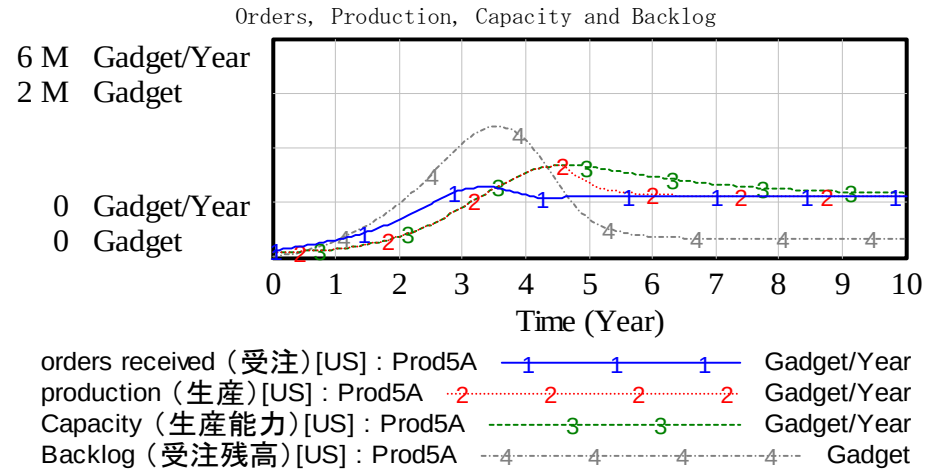
修正後の方程式には、下添え字がすべての変数に対して与えられているものと、一部の変数に下添え字付き変数が与えられている式の両方があります。1つの生産セクターを見て行くことと、すべての生産セクターを見てゆくことをつなぐために良く用いられる関数にSUM関数があります。

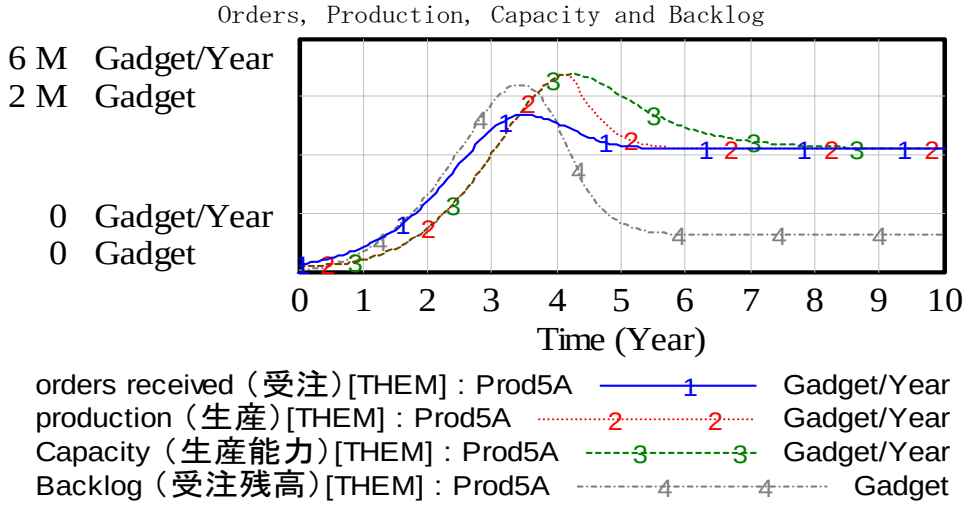
生産セクターでは、orders receivedの方程式を、次に示す様に書き直す必要があります。

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

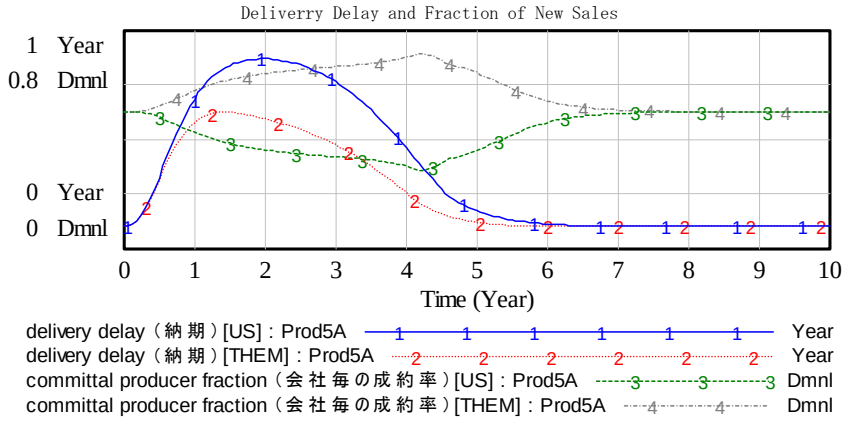
需要を割り付けるためのロジックは、消費セクターで行われます。ここで、方程式が生産セクターの数に依存しないということが重要です。このモデルは生産者の下添え字の範囲定義を変更するだけで、追加した生産セクターを含めるための修正することができます。

このモデルをベースとなる元のパラメータでシミュレートした場合、各会社の相対的な魅力は同じなので、振る舞いは前のモデルと全く同じになります。しかし、time to adjust capacity[US]を2に変更すると、劇的に結果が変わります。





ここで、長期の変化を見るために10年間のシミュレーションを実行しました。生産能力調整期間を長くすることは、調整プロセスをスムーズにすることにはあまり関係ありません。それよりも、他のサプライヤへの需要のシフトが起こり、他のサプライヤは前回の結果の2倍のマーケットを獲得しているのです。この場合、2つのサプライヤのdelivery delay および commitmental producer fractionを見れば、次の様になります。



納期になかなか差が縮まらない違いが発生しています。マーケットシェアをどれくらいの間失い続けるかは興味ある問題です。後に納期が短くなっても、マーケットシェアを取り返すことはできません。THEMは、USよりも迅速に成長するため、より短い納期による取引が市場で増加するため、平均納期も急激に低下します。

6.4 結論

2社以上のメーカーが存在するケースを扱うために、製品の需要と生産能力のモデルを拡張しました。下添え字機能を使って、新製品市場の性質について役に立つ洞察ができました。下添え字機能は、この様な問題を取り扱う際に、優れた技術的バックグラウンドとなります。独占状態においては、キャパの余剰が生じない様に、キャパの増強を平準化することが良いポリシーでしたが、この章で考察した様な競争相手のサプライヤがいるときには、単に競争相手を利するだけという結果になる場合があります。実際に、1つのプロセスの中に含まれる集団に関するモデルをより詳細にすればするほど、新しい政策がインプリメントされたときに、特定の集団がその他の集団より有利にできるものを見つけることができるのです。

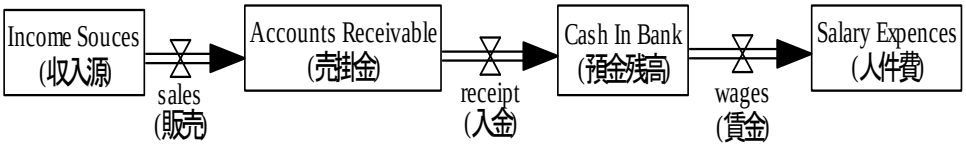
第７章
ファイナンスモデリングとリスク

7.1はじめに

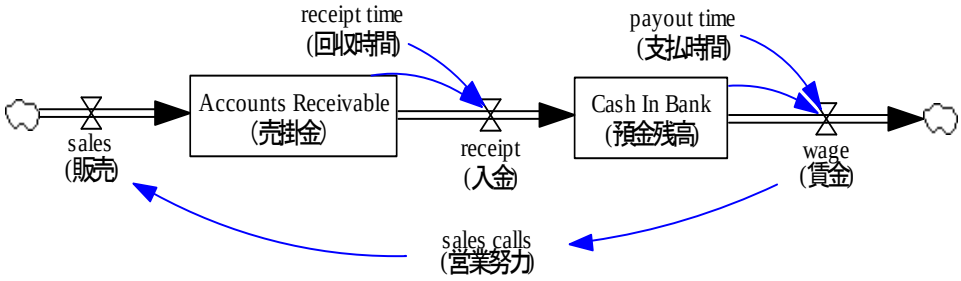
ビジネスモデリングにおいても、標準的な財務指標を使って結果をレポートすることは良くあります。マネージャー自身の経験と結び付きやすいモデルで考えることにより、色々なビジネス上の意図を理解できるからです。本章では、比較的単純な投資案件評価用のファイナンス・モデルを開発し、次にこのファイナンス・モデルを5章で開発した市場成長モデルとリンクします。新規投資はリスクを伴う活動です。構築したモデルにリスク査定を明示的に組込んでいませんが、生み出されるダイナミックスの性質に注目することでリスク査定も可能になります。多くの場合、モデルが生み出しうる可能性がある振る舞いの範囲を理解しておくことはリスクの査定に大変役に立ちます。本章では、財務上のリスクを評価するためにVensimで利用できる多変数感度分析ツールを使用します。

7.2会計と因果関係

ストックとフローによる表現と、標準的な会計実務はいくつかの基本的な一貫性を持っている点で共通しています。処理はフローと捉えることができますし、フローはストックを変化させます。会計では、個々に発生した取引や取引全体を追跡することに焦点を当てます。単純な会社の会計システムを次の様に表すことができます。



フローには因果関係を示す矢印が付されていないことに注意してください。会計システムの目的は、フローを生み出す構造をモデル化することではなく、処理を管理することにあるからです。つまり、処理の原因は会計制度の領域の外にあると言えます。更に、上記の表現で雲（ソースやシンクを表わす）を示す記号がないことに注意してください。これは複式簿記の根本的な考え方として、すべての借方が関連する貸方と結び付けられるからです。また、処理はすべて貸方と借方でバランスが保たれなければなりません。「収入源」と「販売費」は、発生したことの履歴を知ることができますが、それ自体は会社の一部ではありません。より機能を表現しているモデルは次の様になります。



上記のモデルで気づくことがあります。最後のストックを雲に置き換えました。そして、まだ不完全ではありますが、ポリシーを表すための変数を適所に定義しました。そして、それに基づいて、フローを決定しています。この例におけるポリシーとは、①会社にとっての投資回収期間、②顧客にとって納期、そして収入を得るという意味で、③販売員にとっての“売り込みの回数として反映されます。尤も3つ目は他の2つよりはポリシーとして成立するかどうかは少々疑わしいですが。

7.2.1　ストックの詳細

一般に、会計システムに詳細部分を付け加えてゆくことは簡単なことです。追加する会計科目を付け加えるだけです。それとは対照的に、ダイナミックなモデルに詳細を付け加えるためには、ストックを追加することや、対応するフローを付け加えるだけでは済みません。シミュレーションモデル中のすべてのフローを決定するためのルールを定式化する必要があります。1つのストックおよびフローを加えることで多くの新しいフィードバックループができあがるかもしれません。従って、ダイナミックなファイナンシャルモデルを

開発する際に、会計システムが要求するに任せて会計科目を詳細化しては行けません。どこまで詳細にするかはモデルの目的によって決める様にします。

　その構造によってダイナミックスが実現するか否かということで、どこまで詳細化するかを決めるべきです。例えば、製品を売る場合には、商品代金と送料の両方を請求します。会計システムに従って考えていれば、2つの売上勘定科目を別々に設定しようと考えます。しかしながら、投資評価の問題を扱ううえでは、これらを区別することはあまり意味がありません。製品が販売されれば必ず出荷されます。そして販売した商品の代金を受領する際に送料を受け取ります。発送する商品の割合が激しく変化したりしないのであれば、送料をまとめて計上してしまったり、ダイナミックスを変更しないのであれば、製品の売上高としてひとかたまりにして取り扱うべきでしょう。

7.3投資評価モデル

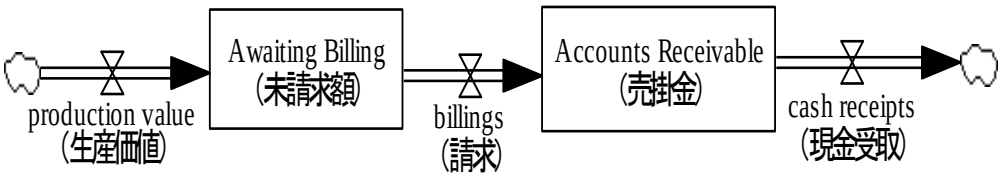
いま*"Thneeds"*という生産設備への投資を考慮しているものとします。そして、Thneedsは注文生産であるとします。(Thneedsは物語から引っ張った架空の商品です。) あなたは設備の生産能力について良く知っていますし、想定される製造原価、生産設備の設置に要する期間、Thneedsの価格、生産変動費、および銀行金利についても分かっているものとします。あなたが問われているのは投資すべきか否かということです。これは、今まで考えてきた問題とはかなり異なっています。すなわち、いかなる構造がその様な振る舞いを生むのかということを考えようとしているのではなく、比較的単純な仮定の結果を調べようとしているのです。モデルのファイナンシャル部分を構築することは、直線的かつ機械的な匂いのする仕事ではありますが、概念上の難しさはあまりありません。一度、ファイナンシャル部分について定式化ができれば、比較的容易に同じ様にファイナンシャル部分を他のモデルに統合することができます。

7.3.1 販売と回収

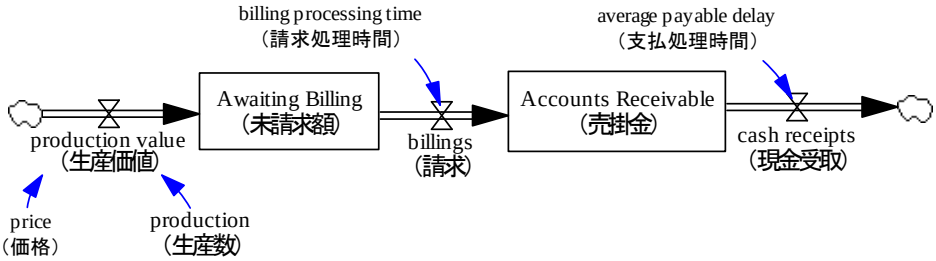
収入と利益の簡単な定式化は次の様になります。

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

この定式化はビッグピクチャーを展望するということでは、大変意味がありますが、財務パフォーマンスの課題を考えるには必ずしも適切だとは限りません。これは、利益とキャッシュ・フローの差に関心があるかという問題であるとも言えます。上記のように定式化するということは、利益とキャッシュ・フローが同じであると暗黙に仮定しています。キャッシュ・フローをより注意深く見るためには、販売や集金のプロセスをより注意深く見ることが必要になります。ストックとフローで表せば次の様になります。



このモデルでは、「生産」がなされると「請求プロセス」が開始することになります。そして請求プロセスが開始すれば、請求がなされ、その請求に基づいて支払いを受領します。「請求」と「現金受取」の定式化は単純です。それぞれ、未請求額と売掛金に対して一定の時定数を乗じて算出されるものとします。



billingsの式は次の様になります。

billings = Awaiting Billing/billing processing time

そして、cash receiptsに対する方程式も同じ様になります。

7.3.2 均衡条件で初期化する

このモデルの定式化にはテクニックとして重要なものが含まれています。ストックの初期化を均衡状態における定常値で行うことです。均衡状態ではストックの流入と流出は等しくなっていると考えられます。実際にどの様になっているのかを理解するために、仮に価格と生産数が両方とも長い期間の間一定だったとします。「生産価値」が「請求」よりも大きい場合「未請求額」は大きくなって行きます。そして「未請求額」が大きくなると「請求」も大きくなります。「請求」は「生産価値」と等しくなるまで上昇し、「未請求額」の変化が止まります。入力が一定の場合、上記の構造は、「請求」と、そして同じ様な議論を経て「現金受領」もまた一定となります。後ろから順番に考えて行くと、現金受取のアウトプットが一定であれば「請求」とインフロー「生産価値」が等しくなければならないのです。

billings = production value

このことが持続するために、「請求」の方程式は次の様になります。:

Awaiting Billing/billing processing time = production value

または、

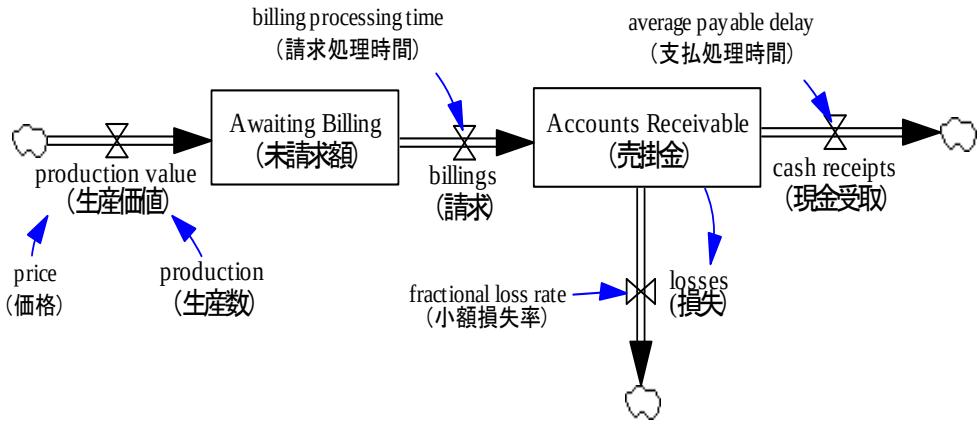
Awaiting Billing = production value * billing processing time

私たちは、この式を「未請求額」の初期化として使います。同じロジックで次の様になります。

Accounts Receivable = billings * average payable delay

Vensimが「売掛金」の値を計算するときに、「請求」、「未請求額」、「生産価値」を計算する必要が出てきます。そして、「価格」と「生産数」も必要になります。Vensimは自動的に順次計算を実行し変則があれば通知されます。

上記の定式化では金銭が受領されれば「売掛金」は減少します。現金受取可能な分だけ売掛金が減少する様に定式化します。これは不幸なことなのですが何らかの理由で代金支払われないことも起こります。これらを考慮の対象外とする必要がある場合もありますが、この問題を処理するために損失のアウトフローを加えることもできます。



ここで、

losses = Accounts Receivable * fractional loss rate

損失を追加すると、「売掛金」の初期値の設定が多少複雑になります。私たちは、「請求」を「現金受取+損失」と等しくする必要があります。すなわち、次の様に定式化できます。

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

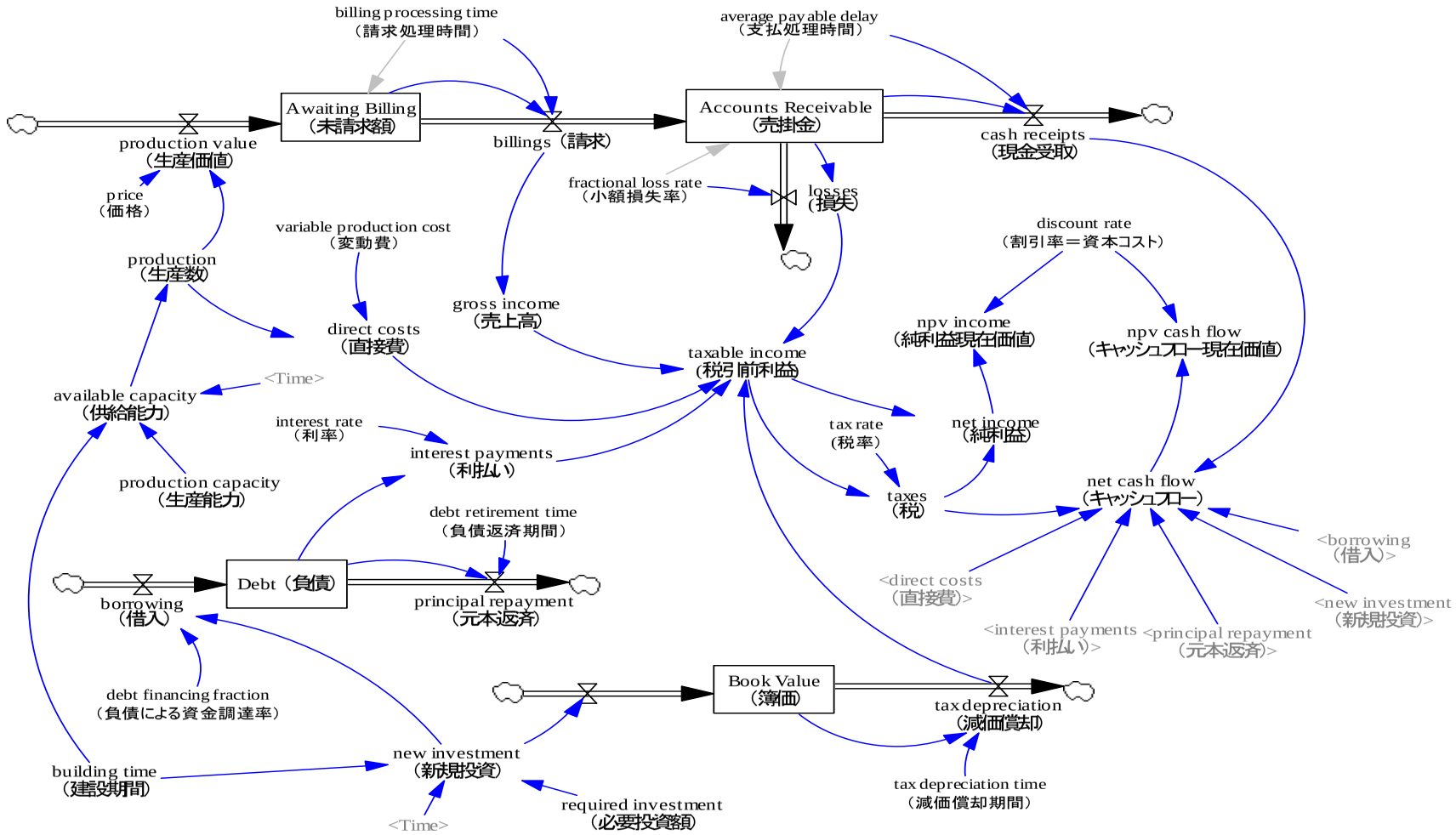
ここで損失率は時定数の逆数になります。

均衡状態でモデルを初期化することは必ずしも必要ではありませんが大変有効となる場合もあります。シミュレーションの初期の部分では、ストックとフローが不均衡となっており、その不均衡状態から均衡状態に抜け出すための調節が優勢となりダイナミックスを支配することが良くあります。傾向として、この不均衡によってもたらされるダイナミックスはシステムに本質的な重要性は持ちませんし、下手をすればモデルの重要なダイナミックスの理解を妨げることになります。今回のモデルについて言えば、実際には時刻0で生産をスタートするのですから、時刻0における2つのストックの値を0にセットすることもできたでしょう。しかしながら、こうすることにより、より一般的にすることができ、いま開発している構造は、他のモデルの中でも容易用いることができます。

7.3.3 完全なモデル(financ01.mdl)

残りの部分は機械的に構築できます。通常は、企業への課税に関する表現は最も難しいものの1つです。ここでは、シンプルかつ多くの状況で適切な表現として、課税が利益に比例する例を選びました。また計算には、定率の減価償却や定率の債務返済を行うものとします。(図表は次頁参照)

ここでストックを2つ加えました。Debt(負債)とBook Value(簿価)です。このモデル中でフィードバックとして、各ストックから各ストックのアウトフローにフィードバックされる短いループがあります。これは、cashflow(キャッシュ・フロー)やtaxable income(税引前利益)に関するかなり複雑な事項を、ダイナミックスの観点でシンプルにモデル化したものです。また、「投資必要額」は「建設時間」にわたって均等に発生するものとし、生産は建設期間が終了する時点で一気に増加するものとします。生産によって実現したキャッシュ・フローを、先行投資に充てる様にモデル構築しています。　モデルは、「利益」と「キャッシュ・フロー」の両方について正味現在価値を計算するためにNPV関数を使用します。NPV関数は、引数として金銭支払いのフロー、割引率、初期値および調整係数をとるダイナミック関数です。　モデルでは初期値として0、調節計数として1を使用しています。これらの引数を指定することによって、NPV関数は、シミュレーションの最終時点で、シミュレーションを開始した時点における現在価値をレポートします。


billing processing time
（請求処理時間）
average payable delay
（支払処理時間）
Awaiting Billing
（未請求額）
Accounts Receivable
（売掛金）
production value
（生産価値）
billings（請求）
cash receipts
（現金受取）
price
（価格）
fractional loss rate
（小額損失率）
losses
（損失）
variable production cost
（変動費）
discount rate
（割引率＝資本コスト）
production
（生産数）
gross income
（売上高）
direct costs
（直接費）
npv income
（純利益現在価値）
npv cash flow
（キャッシュフロー現在価値）
taxable income
（税引前利益）
<Time>
available capacity
（供給能力）
interest rate
（利率）
tax rate
（税率）
net income
（純利益）
interest payments
（利払い）
production capacity
（生産能力）
taxes
（税）
net cash flow
（キャッシュフロー）
debt retirement time
（負債返済期間）
<borrowing
（借入）>
<direct costs
（直接費）>
Debt（負債）
borrowing
（借入）
principal repayment
（元本返済）
<new investment
（新規投資）>
<interest payments
（利払い）>
<principal repayment
（元本返済）>
Book Value
（簿価）
tax depreciation
（減価償却）
debt financing fraction
（負債による資金調達率）
new investment
（新規投資）
building time
（建設期間）
tax depreciation time
（減価償却期間）
required investment
（必要投資額）
<Time>

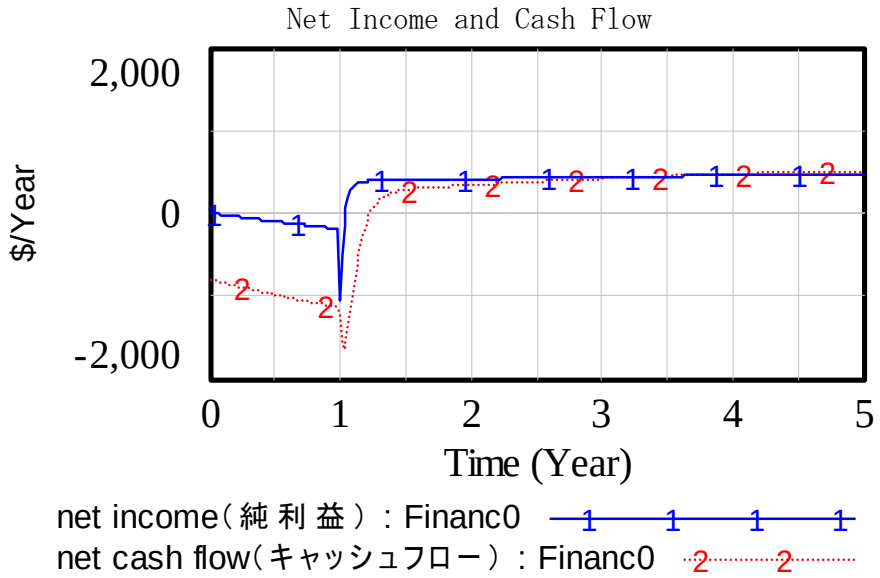
7.3.4 モデルの方程式

```
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
```

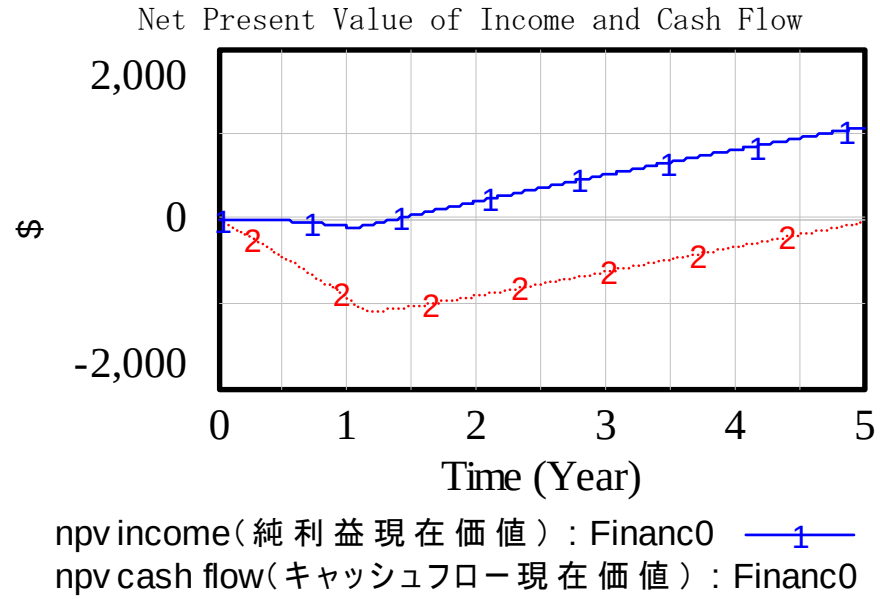
```
fractional loss rate = 0.06
Units: 1/Year
gross income = billings
Units: $/Year
interest payments = Debt * interest rate
Units: $/Year
interest rate = 0.12
Units: 1/Year
losses = Accounts Receivable * fractional loss rate
Units: $/Year
net cash flow = cash receipts + borrowing - new investment -
 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
```

7.3.5 シミュレーション結果

TIME STEP=0.015625で、シミュレーション期間を5年間としてモデルを実行します。



シミュレーションの終了時点で、「純利益現在価値」は1,073ドルとなり、「キャッシュ・フロー現在価値」は-54.70ドルとなります。 これは2,000ドルの総投資回収途上にあるということを意味します。シミュレーションの期間中の現在価値をプロットすれば、次の様になります。



利益とキャッシュ・フローの外見には重要な違いがあります。これは、利益の計算方法がキャッシュ・フローとは異なることから来る結果です。

7.4 感度分析

上記のシミュレーションによると、利益ベースでは、投資は魅力的であると結論を下すことができ一方でキャッシュ・フローベースではそう言う訳には行きません。しかしながら、私たちは、相当多くの仮定に基づくモデルを構築しました。そして、これらの仮定は何れも不確かであることが分かっています。　私たちは1つ1つ仮定を変更したうえでシミュレーションを繰り返し、その意味を確かめることができます。　これに代わるものとして、モンテカルロ・シュミレーションあるいは多変数の感度シミュレーション(MVSS)があります。私たちは不確かな仮定に対しては、その範囲を指定してこの機能を利用することができます。そして、Vensimは不確かな仮定に対しては値をランダムに選択して、モデルを複数回シミュレートします。シミュレーション・コントロールを開始して、感度コントロール・ファイル(例えばfinanc01.vsc)を作成してください。次の様に、パラメータ、分散および範囲(詳細はチュートリアルの11章参照)を選択してください。

```
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)
```

次に、Multivariate(多変数)のオプションを選んで、シミュレーションの数を200とセットしてください。感度分析を行う際には、結果を参照するための変数を1つ選択する必要があります。Vensimは通常、すべての変数の計算結果を保存する様にできていますが、感度分析でその様なことは実際的ではありません。従って、後で結果を参照したい変数のリストを指定します。ユーザーズマニュアルの11章にはパラメータのセット・アップについての情報があります。このモデルについては、次のリストを保存すると良いでしょう。

```
(financ01.lst)
npv cash flow
npv income
net cash flow
net income
```

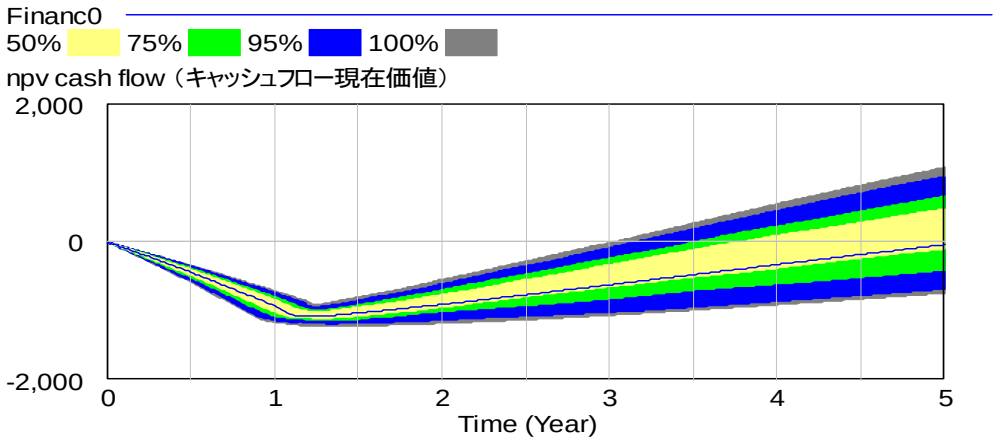
次にシミュレーションを実行します。Vensimは最初に通常のシミュレーションを実行しそして次に多数の感度分析シミュレーションを行います。133MHzのPentiumベースのコンピュータでは、このモデルが、200回のシミュレーションを実行するために数秒かかります。Vensimの構成に依存しますし、また、コンピュータがプロセスをいくつとることができるかによっても変わります。感度分析シミュレーションが始まってしまえば、それが終了されるとともに、ウィンドウに各シミュレーション結果を表示します。万一、遅すぎる場合にはCancelボタンをクリックすることにより200のシミュレーションをすべてが完了する前に、シミュレーションを止めることができます。初期に止めた場合は、結果はここで示しているのとは少し異なるものになるでしょう。

7.5解析結果を表示する

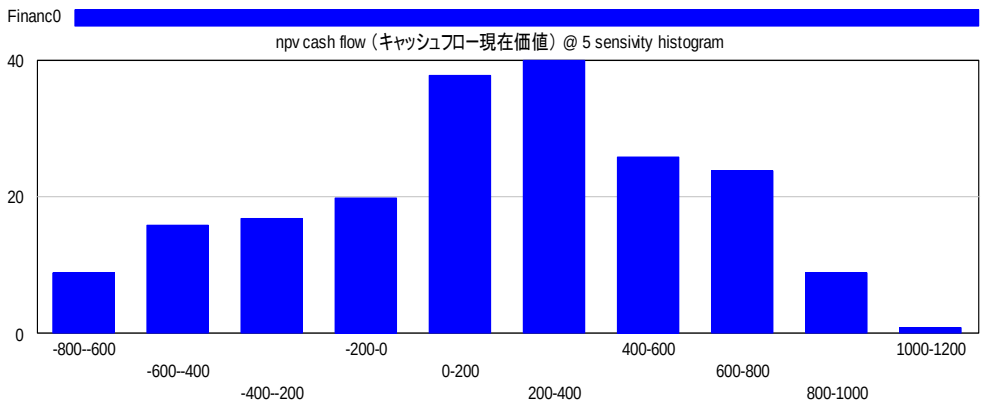
感度結果を表示するために利用できるツールは3つあります。それは、「感度グラフツール」、「棒グラフツール」であり、もし、VensimProfessionalあるいはDSSを使っている場合は「統計ツール」です。チュートリアルの11章は、利用できる感度ツールを備えたツール・セットを導入する方法について記述します。また、リファレンスマニュアルの12章と13章はセット・アップ方法と解析ツールのコンフィギュレーション方法を説明しています。

ワークベンチで変数「キャッシュ・フロー現在価値」を選択して、感度グラフツールをクリックしてください。次の様になるはずです。

このグラフは、時間軸に沿って、キャッシュ・フローベースの現在価値の不確実性を示します。常にシミュレーションの半分は「50%領域」内に収まる値となる、あるいは4分の3が「75%領域」内に収まる値となるといったことを示しています。このグラフに関して注意することが2つあります。一つは、時間が経つにつれて、不確実性が増加することです。これは、現在価値計算は累積的な性質を持っていることに由来する自然な結果です。2つ目は、各パーセンテージは均等にはなりません。棒グラフツールによってこのことははっきり見ることができます。



まず、棒グラフオプションの中にあるヒストグラムと感度のチェックボックスをチェックしてください。そして次に棒グラフツールをクリックしてください。そうすれば、次の様な出力を得るでしょう。



不確実性のパターンとしてUNIFORM(フラット)分布を使用したとしても、「キャッシュ・フロー現在価値」は正規曲線で分布します。これは、モデル中に多くの独立した誤差の発生源があり、それらの組み合わせが結果となっているからです。統計的には、その様な場合は正規分布するということは良く知られています。

これらのことから、次の様な感度分析の使い方もあることが分かります。すなわち、現実に起こりそうな結果あったとします。その様な結果が、どの様な不確実性に基づいて発生するのかということを理解するために感度分析を利用することができます。投資に関する仮説の不確実性が、不確実性を持った結果に帰着することを示します。投資においては、選択オプションに関する判断も必要です。投資判断を下すにあたり、上に示された様なヒストグラムからは、単なる現在価値よりも、本質的に多くの情報を得ることができます。

7.6 金融モデリングと市場成長(financ02.mdl)

ファイナンス･モデルを開発することで、投資評価の支援を目指してきました。しかし、ファイナンス・モデルの構造はそれ自体全く一般的なもので、他にも適用できます。だから、ファイナンス・モデルを4章(prod3.mdl)において開発した市場成長モデルに、容易に適用することができます。そのモデルは、既に投資と生産の概念が含まれています。ファイナンス・モデルと、これら2つの概念を統合することにします。

2つのモデルを連結するために、ファイナンス・モデルからスタートします。モデル構造全体を選択して、それをコピーしてください。市場モデル(prod3.mdl)を開き、新しいビューを作ります。ブランクスクリーン（新しいビュー画面）では、市場モデルにファイナンス・モデルを挿入するためにコピーコマンドの中で「構造の貼り付け」オプションを使用してください。そして、financ02.mdlあるいは別名として新しいモデルを保存してください。Vensimチュートリアルの8章は、別のモデルの中への1つのモデルからの構造を貼るための方法を説明していますので、コピーに関しての詳細はそちらをご覧下さい。

新しいモデル中のビューを見てください。「生産」が「生産0」とリネームされた以外は、同じ様なモデルになる様に処理されます。これは、変数「生産」が市場成長モデルに既に存在しているからです。生産に関する異なる定義を持った新しい構造が、どこからかコピーされて、モデルにペーストされる場合に、矛盾が生ずるのを防ぐためにリネームされます。しかしながら、この場合私たちは古い方の変数である「生産」を使用したいのです。

生産0より下の市場モデルの変数「生産」のコピーを挿入するためにシャドウ変数ツールを使用します。すると、変数「生産」に鍵括弧<>が付された形で表示されます。ここで、変数マージツールを使って、変数「生産0」の上に、変数「生産」をドラッグします。すると『変数「生産0」を消去して「生産」で置き換えますか？』と質問されます。

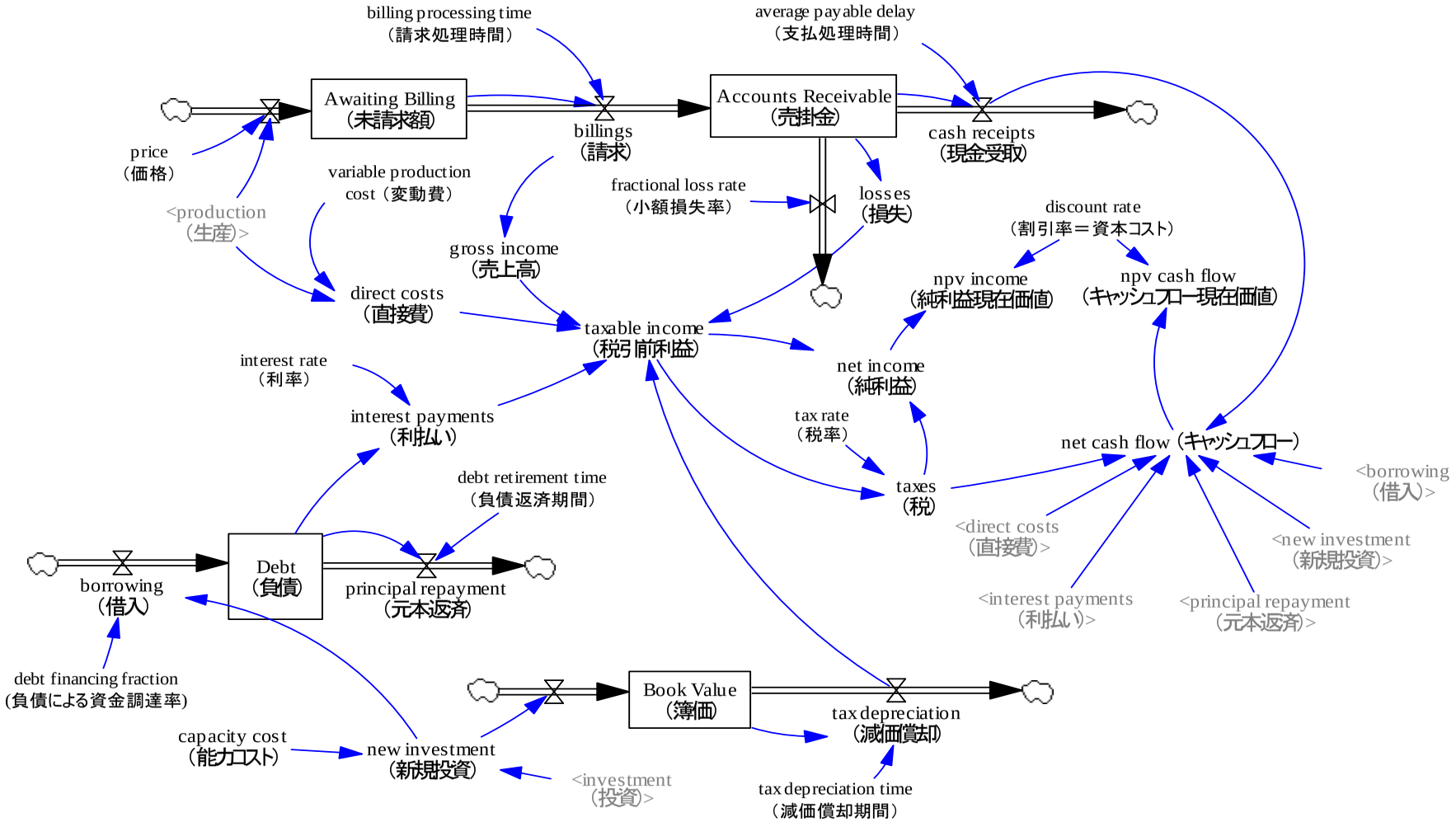
「生産」を「生産0」の上にドラッグするということを間違えないでください。その反対ではありません。もし、ミスを犯したのならば答えは、確認のダイアログでNOとします。正しければ、Yesをクリックしてください。すると、変数「生産0」は「<生産>」で置き換えられます。ここで、Deleteツールを使用し、スケッチのより右下にある「必要投資額」「建設期間」「生産能力」をモデルから取り除いてください。これらの変数はもはや使用されないからです。そして、<Time>から「新規投資」への矢印も取り除いてください。そして、<TIme>を隠しておくこともできますし、そのビューからカットすることもできます。

モデルシャドウ変数ツールをクリックして、「投資」を「新規投資」の近くに付け加えてください。そして、「投資」から「新規投資」に矢印を引いてください。更に「新規投資」の原因として新しい変数「能力コスト」を追加してください。あなたのモデルは次頁の様になるはずです。

方程式編集サブツールをクリックしてください。そうすれば、「新規投資」と「能力コスト」が強調されます。その方程式は次の様になります。

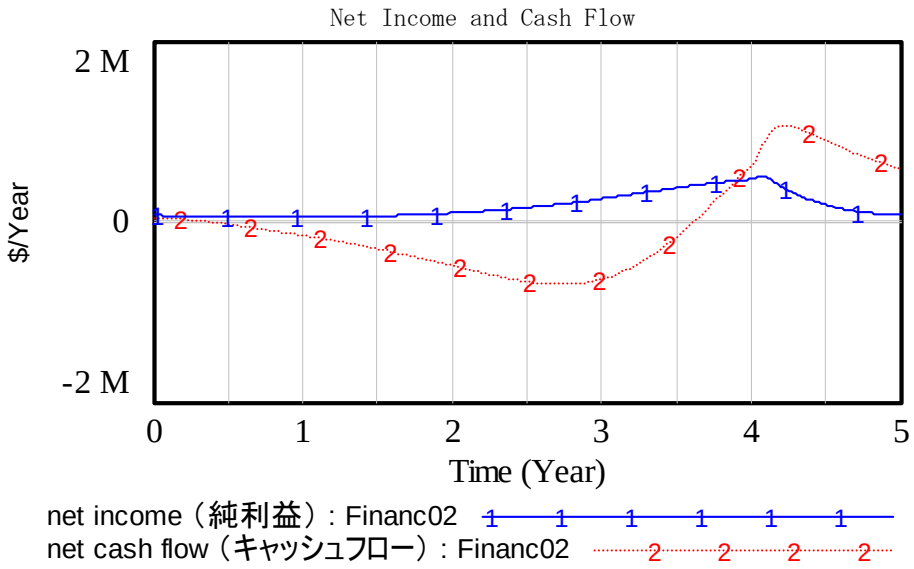
```
new investment = investment * capacity cost
Units: $/Year
capacity cost = 0.5
Units: $/(Gadget/Year)
```

生産能力の陳腐化がより早く進むことを反映するために減価償却期間を2年に変更します。「請求」プロセスで発生するより機微なダイナミックスのためにTIME STEPを0.015625まで変化させます。

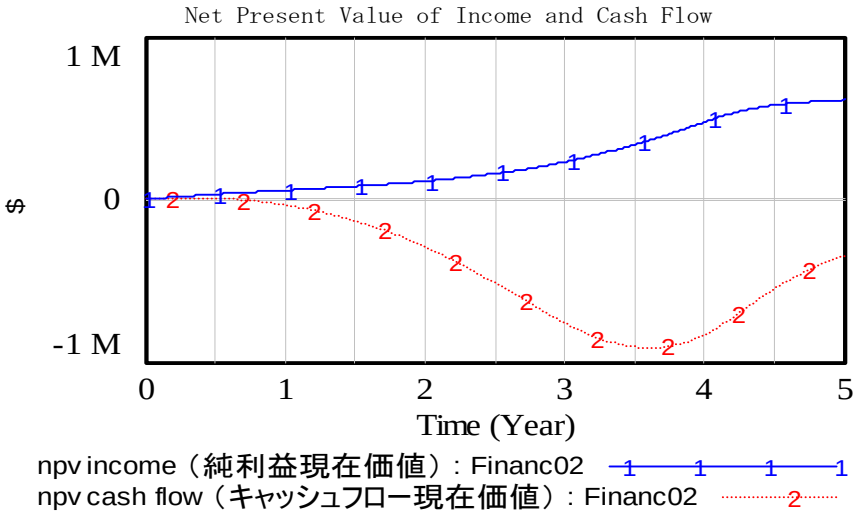

billing processing time
（請求処理時間）
average payable delay
（支払処理時間）
Awaiting Billing
（未請求額）
billings
（請求）
Accounts Receivable
（売掛金）
cash receipts
（現金受取）
price
（価格）
variable production cost（変動費）
fractional loss rate
（小額損失率）
losses
（損失）
<production
（生産）>
gross income
（売上高）
discount rate
（割引率＝資本コスト）
npv income
（純利益現在価値）
npv cash flow
（キャッシュフロー現在価値）
direct costs
（直接費）
taxable income
（税引前利益）
interest rate
（利率）
net income
（純利益）
interest payments
（利払い）
tax rate
（税率）
net cash flow（キャッシュフロー）
taxes
（税）
debt retirement time
（負債返済期間）
<borrowing
（借入）>
<direct costs
（直接費）>
<new investment
（新規投資）>
Debt
（負債）
borrowing
（借入）
principal repayment
（元本返済）
<interest payments
（利払い）>
<principal repayment
（元本返済）>
debt financing fraction
(負債による資金調達率)
Book Value
（簿価）
tax depreciation
（減価償却）
capacity cost
（能力コスト）
new investment
（新規投資）
<investment
（投資）>
tax depreciation time
（減価償却期間）

7.6.1 シミュレーション結果

このモデルのシミュレートは次の様な結果になります。



純利益とキャッシュ・フローでは非常に重要な違いがあります。キャッシュ・フローはスタートアップの間は数年間マイナスとなります。キャッシュ・フローが最も大きくなるところで、純利益は落ちています。現在価値でも相違があります。

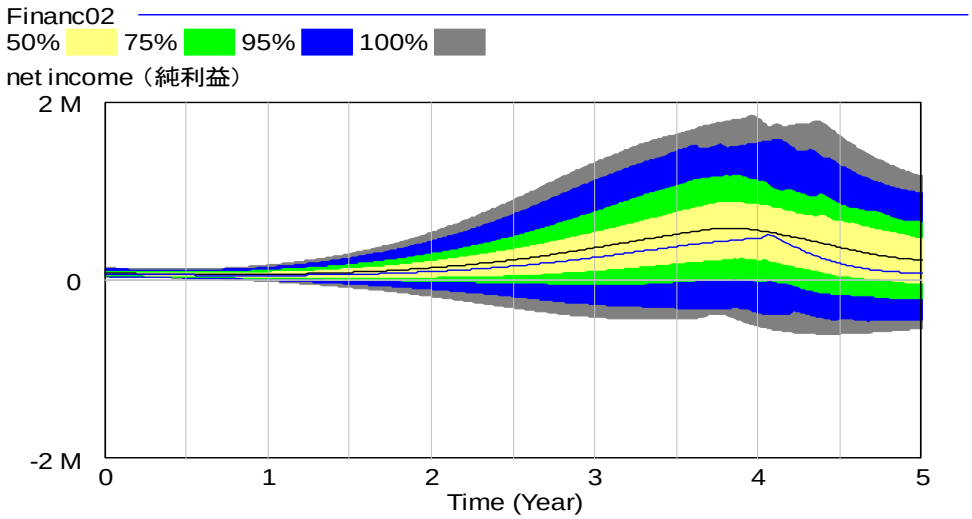


7.6.2 感度分析

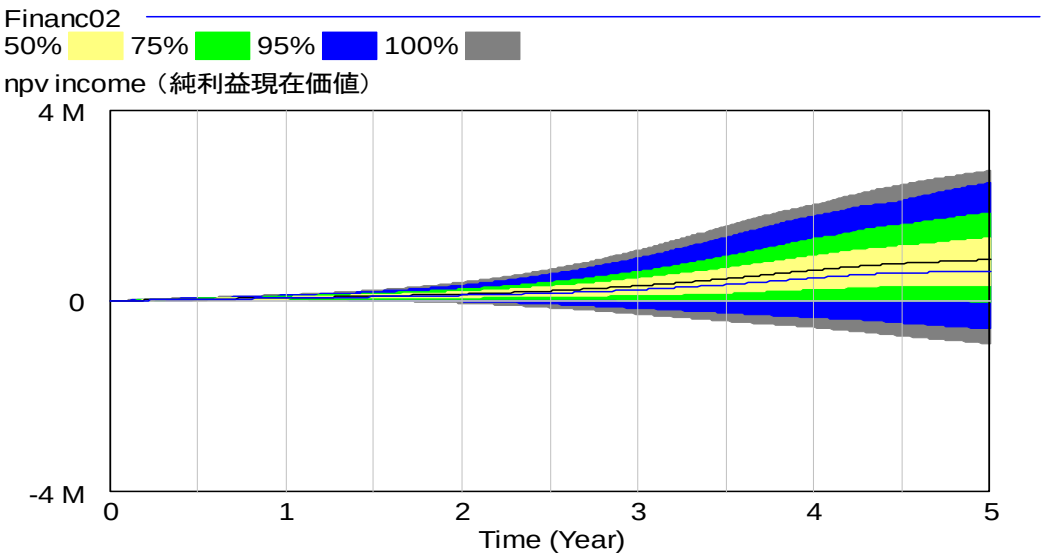
このモデルを使ってセンシティビティ･チェックを実行することができます。その際に、基本コストに関する仮定だけでなく、市場セクターのパラメータについても不確実性を考慮することができます。そのためには、次の様な感度コントロール・ファイル(financ02.vsc)を使用してください。

```
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)
```

これをシミュレートすれば、純利益に関する結果を得ることができます。



ここで、起こりうる結果の範囲は大変大きくなります。同様に利益の現在価値は次の様になります。


Financ02
50%
75%
95%
100%
npv income（純利益現在価値）
4 M
0
-4 M
0
1
2
3
4
5
Time (Year)

7.7 結論

多くの場合、モデル中のファイナンス・セクターは、モデル全体のダイナミックスの重要な部分となることもあるでしょう。キャッシュ・フローと利益、何れをフィードバックするかで人々の振る舞いは変わってきます。ある場合には、新投資を削減するとか、低い利益やキャッシュの残高不足に対応して、メンテナンスの支出を先延ばしすることが必要かもしれません。本章の中で開発されたモデルは、これらのことを考えるための材料になるはずです。本章において、投資計画の財務的な結果を評価し、不確実性を調査するためのモデルを開発しました。開発されたモデルでは、重要なダイナミックスを見出すに至っていませんが、計測ツールとして働くことは確かめることができました。ビジネスを表すモデルに、ファイナンス・セクターを含めておき、モデルに基づいて下された投資の結果と一緒に伝えて行くことは、政策オプションの意味を理解する手段として有用です。多くの場合、モデル中のファイナンス・セクターは、モデル全体のダイナミックスの重要な部分となることもあるでしょう。キャッシュ・フローと利益の何れをフィードバックするかで人々の振る舞いは変わってきます。ある場合には、新投資を削減するとか、低い利益やキャッシュの残高不足に対応して、メンテナンス支出を先延ばしすることが必要かもしれません。本章の中で開発されたモデルは、これらのことを考えるための材料になるはずです。

第８章

振り子および振動

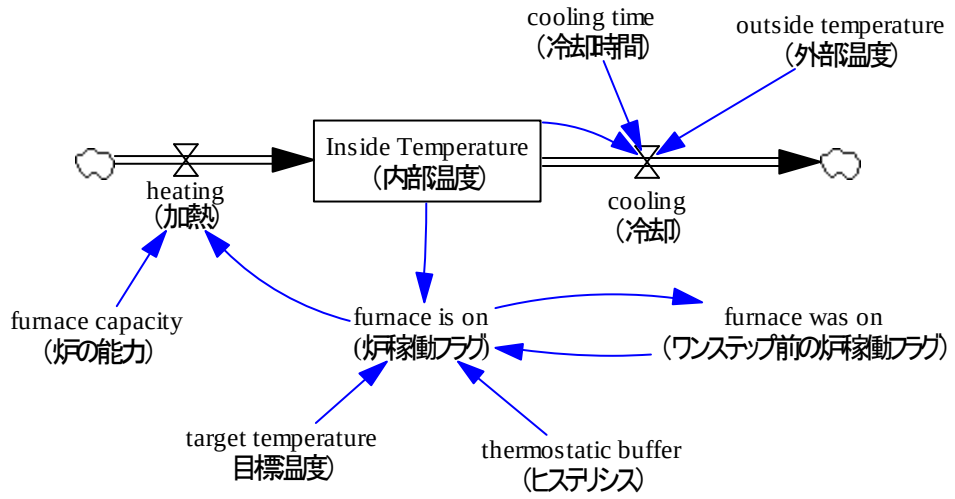
8.1 はじめに

詳細な労働力-在庫モデルを開発しました。そして、労働力-在庫モデルの振る舞いは振動するということを見出しました。このモデルは、労働力と在庫に関する問題を解決するために役立ちますが、振動を引き起こすプロセスを洞察という意味でも重要です。本章では、労働力-在庫モデルと同様の基本構造を持つ異なる問題について考えます。何れのモデルも振動を起こしますし、注意すべき特性があります。

8.2 自動温度制御装置(thrmstat.mdl)

自動温度調整装置は、電気炉等に付けられた単純なメカで実現されていることがよくあります。このシステムは振動させることを目的にしているという意味で興味深いものがあります。サーモスタットの目的は、あまり頻繁にスイッチをオン・オフせずに、室温を快適な範囲内に維持することです。このことを実現するために、制御のためのバッファー温度範囲を設けなければなりません。室内の温度がバッファー範囲の上限まで上がったとき炉はオフになり、バッファー範囲の下限まで温度が下がったときはオンになります。バッファー範囲内の温度では、炉は現状の状態に留まります。すなわち、現状がオンならばオンで、オフならばオフで留まり続けます。

これまで考えた他のモデルと異なる理由は、それが離散量のイベント、つまり炉をオンしたりオフしたりすることを取り扱っているということです。更に問題を難しくしているのは、炉のスイッチをオンにしたりオフにしたりすることは、炉の現在の状態（既にオンかどうか）に依存します。これ(現在の状態が現在の状態を決めるということ)は、パラドックである様に見えます。実のところ、炉のオン・オフのスイッチングは、離散量の事象ではありません。炉を暖めたり、冷ましたりするプロセスに比較すると、スイッチングは瞬間に見えるということです。なので、パラドックスを解決するために、時間の粒度という概念を導入したうえで、目的に沿ったモデル化を行うために、炉が最後(直近)にどの様な状態であったのかということを追跡することによりこのパラドックスを解決します。



モデルの式は次のとおりです。

```
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
```

冷却ための定式化は、労働力-在庫モデルの中で使用される「正味雇用率」の調整の定式化と同じです。「暖房」が0ならば冷却効果が発生し、「温度」は現在の値から「冷却時間」

を経て「外気温度」になります。「冷却時間」の解釈としては、部屋の中の個々の粒子活動が下がるための時間として与えられるのですが、その様な理解はあまりここでは有用ではありません。なぜなら、個々の粒子は分子であり、温度の概念には分子運動の平均という概念が既に含まれているからです。冷却時間とは、現在の室温と、目標温度または平衡温度のギャップのほとんど(63%)が埋められたときとみなされます。例えば、外部の温度が0で、内部の温度が100度であった場合、暖房をオフにして8時間経てば温度は37度になります。この時点では室温は決して0度にはなりません。しかし、24時間後には室温は0度に非常に接近した値(5度)になります。

heating = furnace is on * furnace capacity

Units: Degrees Fahrenheit /Hour

furnace capacity = 10

Units: Degrees Farenheit/Hour

「暖房」は、部屋にエネルギーを追加することを意味します。それは蛇口をあけて流しに水を流し込むのに似ています。室温を目標変動幅内に維持するため暖房をオンにするロジックが組まれています。

```
  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
```

この定式化は、暖房が既にオンであり、温度が(目標+バッファー)を下回っている場合、暖房はオンのままとし、そうでなければ、温度が目標値から制御バッファーの数値を引いた数値未満である場合のみ、暖房はオンされます。

target temperature = 70

Units: Degrees Fahrenheit

thermostatic buffer = 2

Units: Degrees Fahrenheit

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

Units: Dmnl

「ワンステップ前の炉稼働フラグ」は記憶しておく必要があります。そして、それはダイナミック変数でなければなりません。DELAY FIXED機能はこのためのメモリ機能を果します。DELAY FIXED関数が扱える最小の遅れはTIME STEPです。また、遅延時間が0であるということは炉の直近の状態に関心があることを強調するために使用します。「ワンステップ前の炉稼働フラグ」は次の様に定式化することができます。

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

結果は同じになるかもしれませんが、「ワンステップ前の炉稼働フラグ」に0から1以外の値を残すことは、数字的な問題を残します。オイラー積分法以外の積分法が使用される場合は特に問題となります。

「ワンステップ前の炉稼働フラグ」の定式化は常に0からスタートすることに気づくでしょう。もし、「炉稼動フラグ」の値で始めるとしたならば、次の様になります。

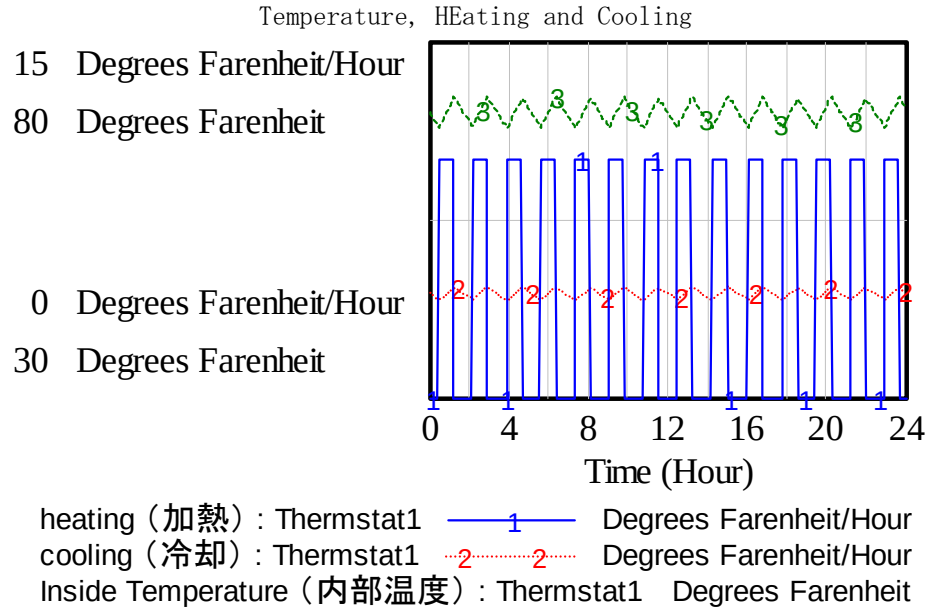
furnace was on = DELAY FIXED(furnace is on,0,furnace is on)

ここで初期値の同時問題が発生します。シミュレーションを実行するために「炉稼働フラグ」に初期値を与えなければなりません。

コントロール定数として、1時間のうちの1/16時間(.0625)の間隔で24時間このモデルを実行する準備ができています。

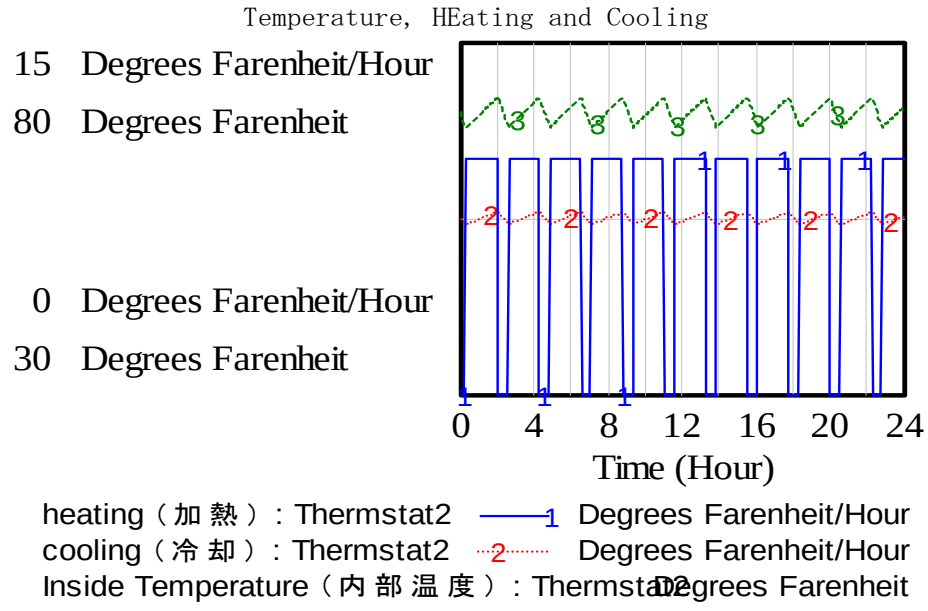
8.2.1 シミュレーション

このモデルをシミュレーションすると、次の様な振る舞いが得られます。



温度のパターンは１日中ぎざぎざになります。フロー「冷却」は、4(Farenheit/時間)の値のまわりで適度に変化します。炉のスイッチは1時間弱オンになり、それより多少長い時間オフになります。

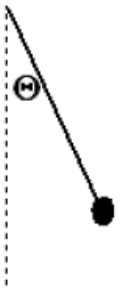
「外部温度」を(35から)10度に下げた場合を実験した場合、次の様な振る舞いになります。



この場合、炉はより頻繁にオンになり加熱され、より急激に冷却されるようになります。

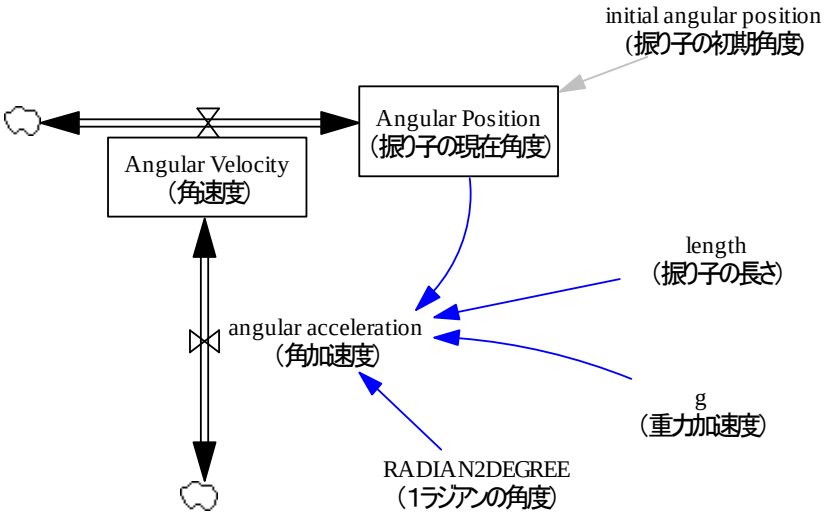
8.3 振り子(pendulum.mdl)

振り子は、ホックやアタッチメントからつるされた糸に錘が付けられたものです。そして皆が良く知っている装置であり、数世紀の間、時計の中で使用されてきました。振り子が時計として価値がある理由は、運動の振幅が小さければ、比較的一定の周期を維持できるからです。



もし、鉛直方向からの振り子の変位をΘで表した場合、振り子に加わる力は、糸の方向と糸に垂直な方向に分解することができます。それらはそれぞれ、mg cos(Θ)とmg sin(Θ)となります。(ここで、m が振り子の質量で、g が重力加速度です。) 錘は糸に垂直な方向にだけを移動させることができるので、加速する力は糸に垂直な方向に作用します。その結果、直線加速度はg sin(Θ)となります。直線加速度を糸の長さで除することにより角加速度に変換することができます。

この物理的な問題をモデルで表すと次の様になります。



カスケード接続されたストックに注目してください。対象を加速することはその速度を変化させます。次に、速度はその対象の位置を変化させます。これは、そのフローそのものもまた、更に1つのストックであることを意味します。この例では道理にかなっていますが、この様な例は物理学以外のモデルでは稀です。労働力-在庫の例の中で、在庫は「振り子の現在角度」と、労働力は「角加速度」と類似しています。しかしながら、労働力は、直接「在庫」へ積みあがりませんが、その代りに、在庫を積むプロセス(生産)の原因とな

っています。そしてその様なことはすべて、1章で議論された様なループをより多く導入することへとつながって行きます。このモデルではフィードバックループが1つだけあります。

単純のために、このモデルは、degree単位で計算された角度を使用します。「ラジアンの角度」は、ラジアンをdegreeに変換し、sin(Θ)の計算の中で使用される定数(=360/2π)です。方程式内の次元の一貫性を維持するために、測定単位ラジアンは、無次元と考えられます。

このモデルは次のとおり定式化できます。

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

この方程式において、「1ラジアンの角度」によって除することにより、角度の位置がdegreeからラジアンに変換されていることに気づくでしょう。
結果(sin(Θ)g/length)はラジアンであり、「1ラジアンの角度」によって乗ずることによりdegreeに変換されます。

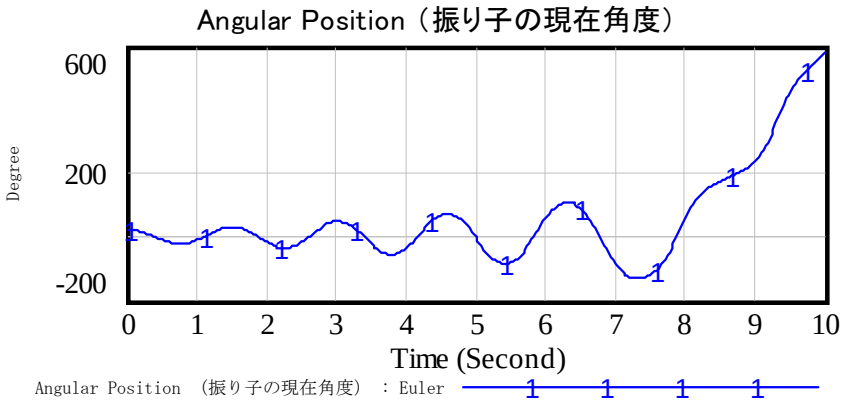
g = 9.2
Units: Meter/Second/Second
initial angular position = 20
Units: Degree
length = 0.5
Units: Meter
radian2degree=57.296
Units: Degree:

8.3.1 振り子のシミュレーション

振り子の運動方程式を既に解いた経験があるならば、上記はとても例外的に見えるかもしれません。仮定の単純化sin(Θ)=Θがなされるのが普通だからです。私たちは、ここでは一般解を求めるのではなく、方程式をシミュレートしているので、単純化は必要ありません。実際、振り子のより大きい振幅を理解するためにこのモデルを使用することができます。もし、単純化してしまえばそれは不可能になってしまいます。

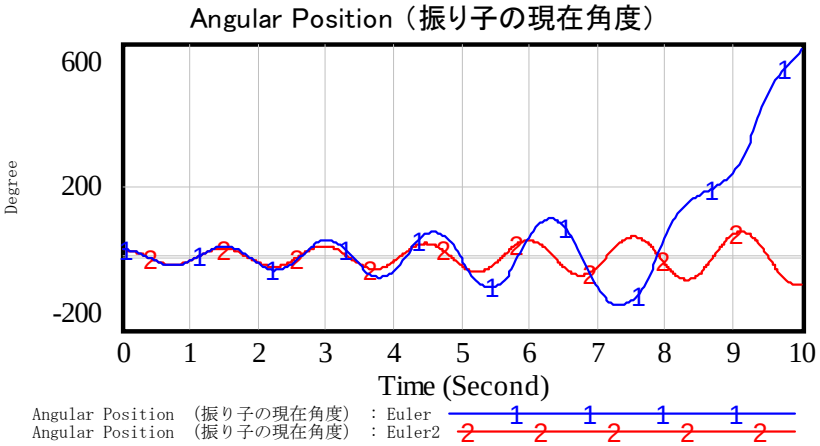
8.3.2 オイラー積分法

TIMESTEPを1秒の1/32、(.03125秒)として10秒間このモデルをオイラー積分法でシミュレートしてみます。すると次に示す様な驚くべき結果になります。



このグラフは20度まで錘を上げて放せば、錘が数回前後に行き来して、次に、その軸のまわりで回り始めることを示しています。子供はこの様なことに大変興味を持つかもしれませんが何かが間違っています。それは積分法と関係しています。

もしオイラー積分法が原因ではないかと疑いを持ったときは、積分の刻みを半分にして結果が変わるかどうか確かめることです。ここでは、TIME STEPを.015625変更して確かめることができます。



結果は大きく変わりましたが、まだ、振動はどんどん大きくなり続けています。恐らく何秒が後には回転が起こってしまうでしょう。だから、このシミュレーション結果はまだ意味をなしているとは言えません。

オイラー積分法を使用して、このモデルから意味をなす結果を得るために、TIME STEPを非常に小さくする(<.001)必要があります。これを試す場合は、SAVPERを.0625または.03125とすることを忘れないでください。さもなければ、Vensimは変数毎に10,000個の値を格納しようとしまい、恐らく失敗してしまうでしょう。このことに注意して実行すれば、より良いシミュレーション結果が得られます。

このモデルにおいてオイラー積分法が上手く行かない理由は、このモデルは、安定と不安定の縁にあり、減衰しないオシレーターを表しているからです。オイラー積分法は単純な線形の外挿法です。曲線を外挿法によって値を推定しようとすると、方向転換する点では常にオーバーシュートしてしまいます。通常は、小さなオーバーシュートはあまり懸念する様な事態には至らないのですが、この場合は各サイクルで少しずつの誤差が積み重なってしまい、完全に違った方向に結果が向かってしまいます。

8.3.3 ルンゲ-クッタ積分法

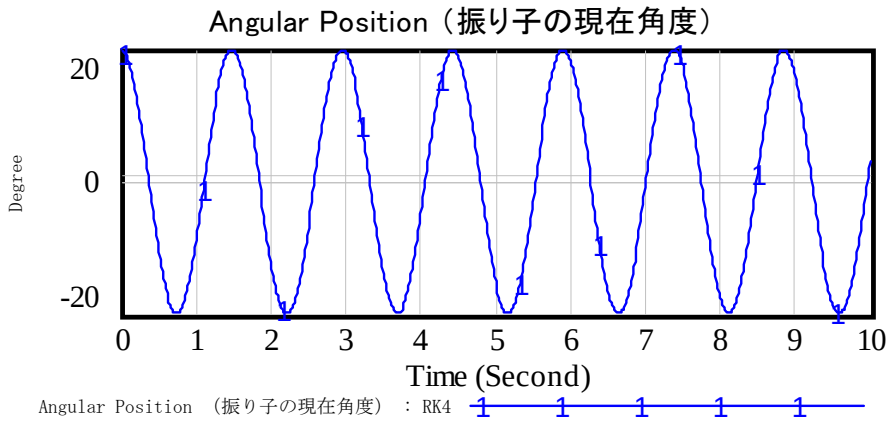
Vensimはこの様なモデルに対処するためにもう一つの積分法である､ルンゲ-クッタ積分法をサポートしています。これは、軌跡の両端に加えて軌跡がどの様に変化しているかを見て、よりよい結果を得ることができる、より高次の外挿法です。

ルンゲ-クッタ積分法では、RK4自動、RK4固定、RK2自動、RK2固定を選択することができます。これらは基礎となる連続時間系システムを、より高次の近似(RK2は2次、RK4は4次)を使用して、シミュレーションしていることを示しています。一般に、RK4自動は最も精度が高い方法です。何故ならばそれは、自動的に、正確さをチェックするために、上(オイラー積分法のところ)で行った様なTIME STEPを減少させるテストを自動的に行います。精度が許容範囲を下回る場合、希望の精度が得られるまで、積分の刻み間隔を減少させます。これはいつも成功するとは限りません。所定の精度を達成できないときは警告メッセージが出ます。精度チェックを行わないという点を除いて、RK4固定はRK4自動と同じです。

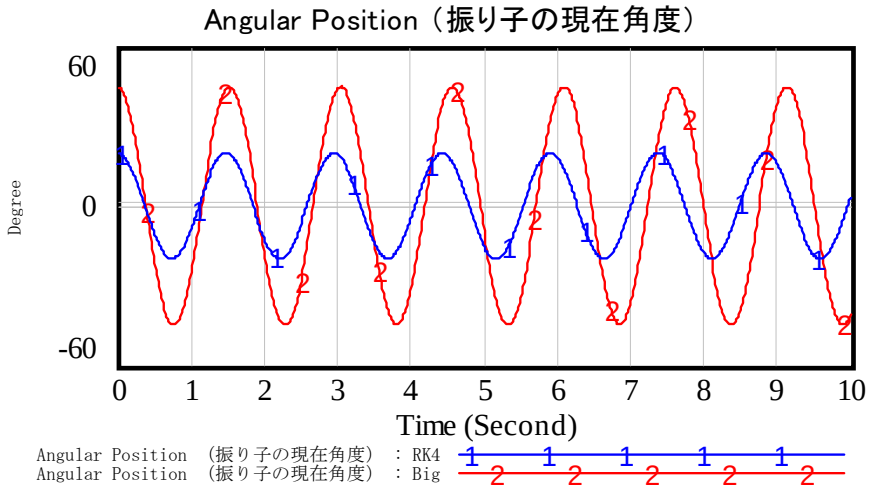
RK4固定はRK4自動よりも実行速度は速くなります。しかし、精度が問題となっていないことを確かめるためにオイラー積分法のときの様に、テストしてみる必要があります。

ルンゲ-クッタ積分法を使用しているときには､すべての中間の計算結果を後で参照することができるとは限らないということは重要です。TIME STEPや通常はSAVEPERも積分法によって影響されません。ルンゲ-クッタ積分法ではTIME STEPとTIME STEPの間においても計算が行われます。従って、レベルがその流入と流出の蓄積の結果であるとは考えられない様な値にあることもあります。連続時間系システムでは、これは重要ではありません。しかし、場合によってはこのことが非常に混乱を起こすかもしれません。

小さなモデルでは､RK4自動を使用することが最も良い選択です。その他の積分法は、正確さとシミュレーション時間のトレードオフを考えなければならない様な場面で、選択されるかも知れないということを意図して提供されています｡先ほどの振り子のモデルをRK4自動を使用してシミュレーションすれば次の様な結果を得ることができます。



これは、プラス・マイナス20度で振動が継続した結果です。45度(これはもはや小さな変位とは言えません)まで最初の位置を変更することができます。



角度が大きくなっても同じ様な振る舞いを観察することができます。しかし、周期は角度を大きくするとより長くなります。これは、物理学入門において典型的に教えられている内容とは異なります。何故なら、物理学では $\sin(\Theta)=\Theta$ という近時をおきます。物理学で教えられていることと同じ内容を観察したいときはモデルを次の様に変更してください。

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

どの様な振る舞いが生じるか確かめてみてください。

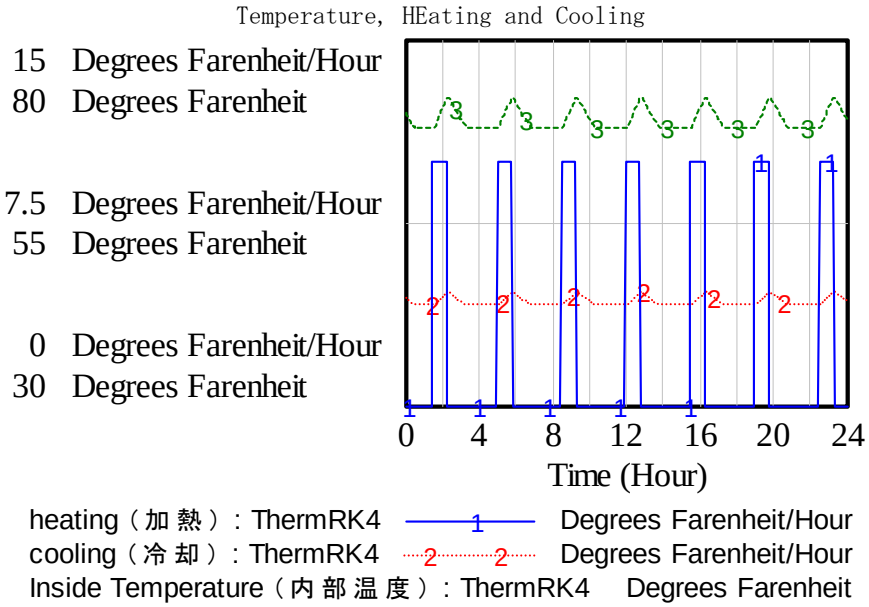
8.3.4 ルンゲ-クッタ積分法の詳細

私たちは本章で2つのモデルを見てきました。そのうちの1つは、意味のある結果を得るためにルンゲ-クッタ積分法が必要でした。しかしながら、サーモスタット・モデルではルンゲ-クッタ積分法を使用しませんでした。それには理由がありました。

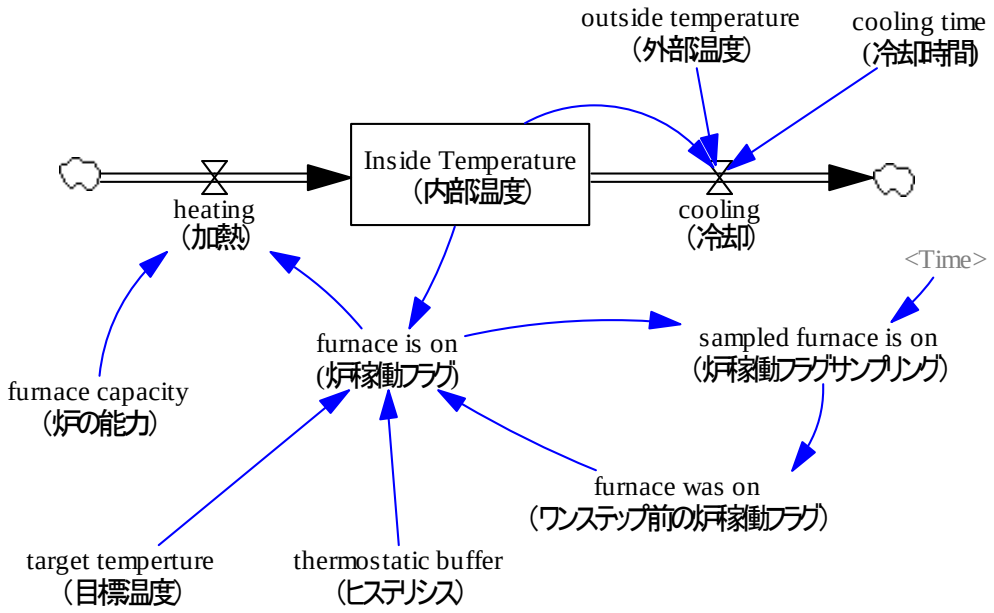
8.4サーモスタット・モデルをルンゲ-クッタ積分法でミュレーションする(thrmstt2.mdl)

サーモスタット・モデルではオイラー積分法を使用しました。もし、RK4自動を使用して、このモデルをシミュレートしようとすれば、次の様なメッセージが出るでしょう。

そして、その結果は次の様になります。



これは明らかに前の結果とは異なります。炉がオンになっていないのに、長い間温度は水平になっています。このモデルは本質的に不連続です。この様なモデルにはルンゲ-クッタ積分法は適用できません。「温度」がフラットになっているのはTIME STEPの間における計算の結果です。何が起こっているのかと言えば、TIME STEPの間における計算で炉をオンにすべきトリガー・ポイントが到達しているのです。しかしながら、モデルの定式化で使っているDELAY FIXEDは離散時間系の遅れであり、離散時間系の遅れはTIME STEPでのみ変化するので、「ワンステップ前の炉稼働フラグ」は0のままになっているのです。



積分法の内部計算を見ることはできません。しかし、私たちは、何が起こっているのかを垣間見ることができるモデルを作ることができます。(thrmst2.vmf) 私たちは新しい変数「炉稼働フラグサンプリング」を導入します。

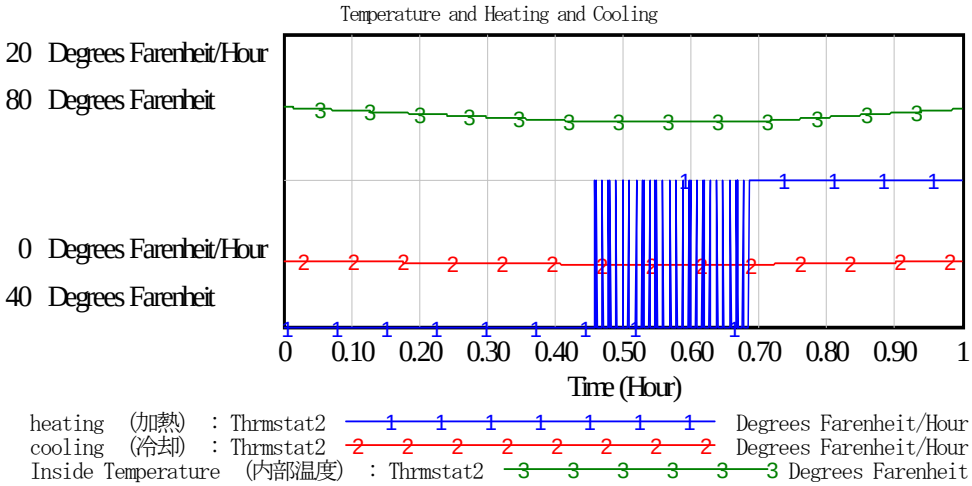
それは、以前のTIME STEPにおける「炉稼働フラグ」と同じ値です。しかし、TIME STEPがより小さい値にしたときは、以前のTIME STEPにおける値が保持されます。これは次の様に定式化できます。

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

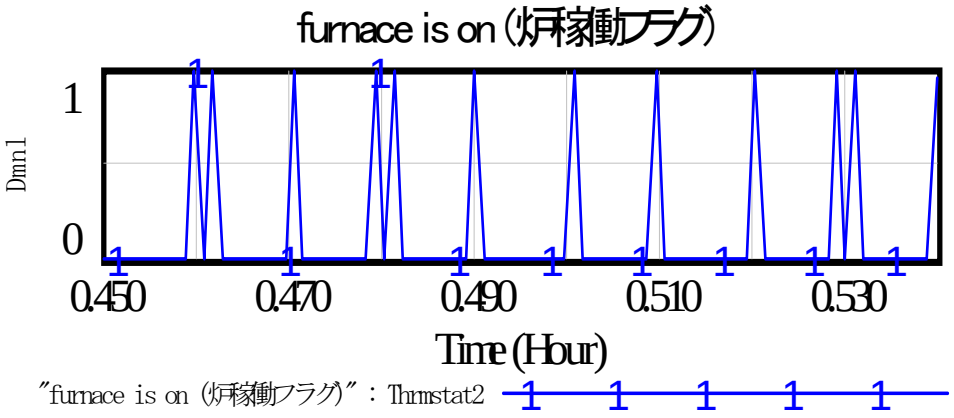
この方程式は、TIMEが0.0625の倍数になるまで、「炉稼働フラグサンプリング」の値を保持します。そして、TIMEが0.0625の倍数の時点で新しい値にスイッチします。「ワンステップ前の炉稼働フラグ」は「炉稼働フラグサンプリング」によって書きかえられます。

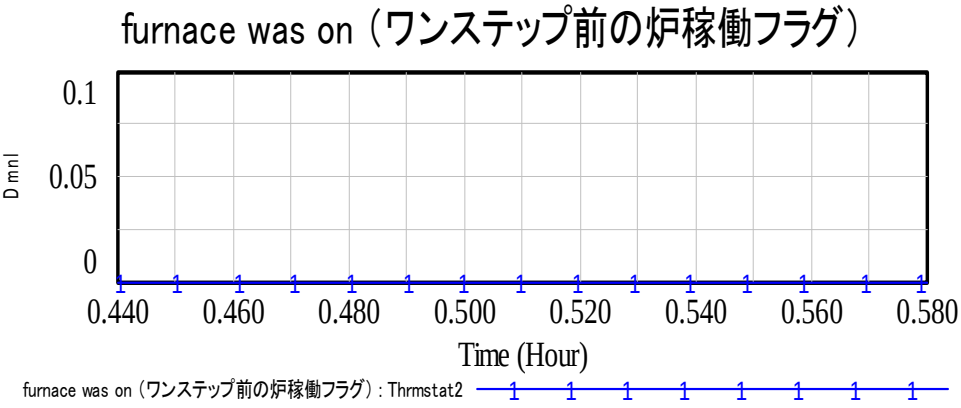
furnace was on = DELAY FIXED(sampled furnace is on, 0, 0)

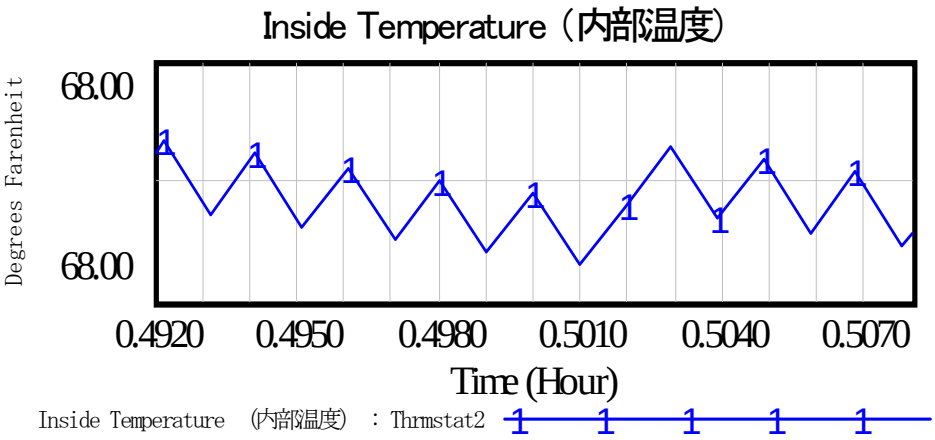
TIME STEPが0.0625の場合、このモデルは最初のモデルとまったく同じ振る舞いをしています。しかしながら、TIME STEPを1/1024(0.0009765625)にセットし、シミュレーション時間の長さを1(更に、OUTSIDE TEMPERATUREを34としました)に縮めれば、次の結果が得られます。



温度が一定になる箇所が炉のオン・オフのサイクルの中に存在します。(黒い長方形となって現れている) この区間を(シフト・キーを下げておきマウスを引くことによって)ズームインすれば「炉稼働フラグ」の動きを見ることができます。







時間スケールを見れば、今、私たちは大変短い期間にフォーカスしていることが理解できます。温度はほとんど一定です。炉は温度を一定に保持するためにオン、オフをスイッチングしています。ワンステップ前の炉稼働フラグはずっと0に留まっています。ワンステップ前の炉稼働フラグが0に留まっている理由は、0.0625の倍数でのみサンプリングして更新されるからです。炉が頻繁にオン・オフしているので、炉をオンにしたり、オフにする条件を探すことに切り替えたりする瞬間を追いかけるには時間がかかりすぎると思うかもしれません。

しかし、私たちが今ここで調査している事柄は物理的なシステムの振る舞いではなく、このモデルをシミュレーションするための技術によってもたらされる振る舞いであることを強調しておくことは重要です。もしバッファーのないサーモスタットを実現しようとしたときに、この課題を物理的なシステムに直接適用できるかも知れません。

8.4.1 積分法に関する結論

オイラーおよびルンゲ-クッタ積分法の結果間の劇的な違いは、それらが本章にあるほど表面化することはめったにありません。社会システムのモデルでは積分法によって結果が劇的に変わることは稀です。物理的なシステムでは、それぞれの要素の相互関係は正確であり、物理的な法則に基づいてルンゲ-クッタ積分法で処理することが望ましい結果につながります。しかし、1つの例外は明示的に離散時間系の概念を導入したサーモスタット・モデルの場合です。離散時間系の概念(特にDELAY機能)を使用した場合は、オイラー積分法によるべきです。

第９章

離散時間系関数

9.1 はじめに

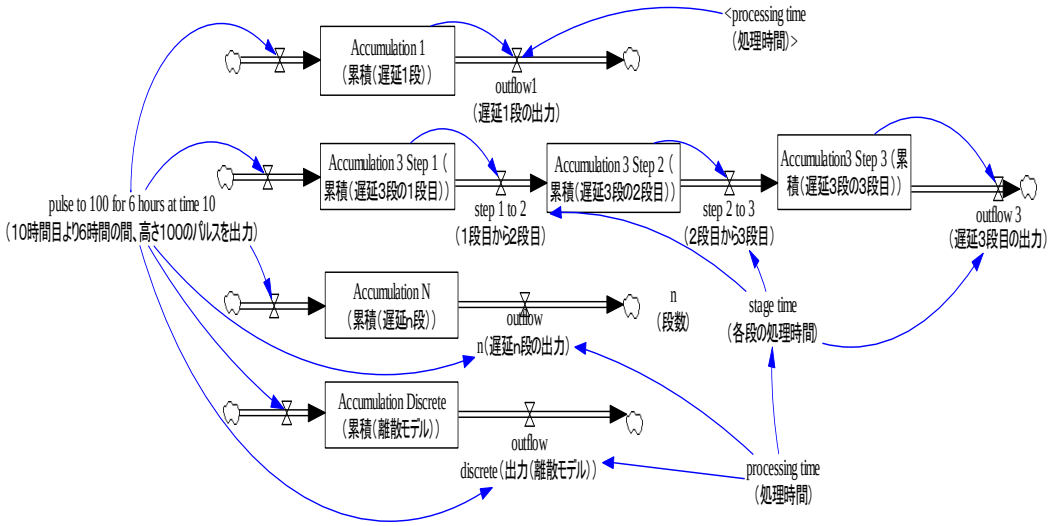
システムダイナミックスモデルは大抵の場合、連続時間系で定式化したストックとフローから構成されます。そして大抵の問題では連続的時間系で扱うことが適切です。しかし、あるケースでは、離散時間系のアプローチが便利な場合もあります。その場合、連続時間系による定式化で容易に近似することもできますが、離散時間系の定式化を使った方が簡単です。Vensimには離散時間系をサポートする多くの機能があります。

この章は、離散時間系のプロセスをVensimの機能を使って表現するための例を多く示します。目的はモデルを構築する際のヒントにするためです。

Vensim PLEで使用できる離散時間系関数はDELAY FIXEDのみです。

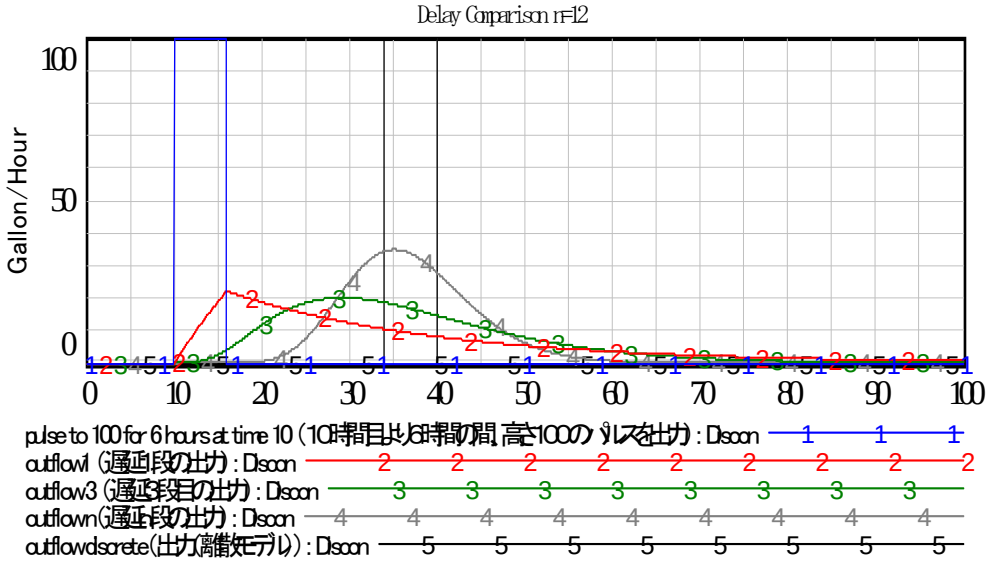
9.2連続時間系と離散時間系の遅れ

離散時間系の例を取り扱う前に、離散時間系と連続時間系のモデルがどの様につながっているかを見ておきましょう。そのために、いくつかのタイプの遅れ(discon.mdl)が混在するモデルを考察してみましょう。

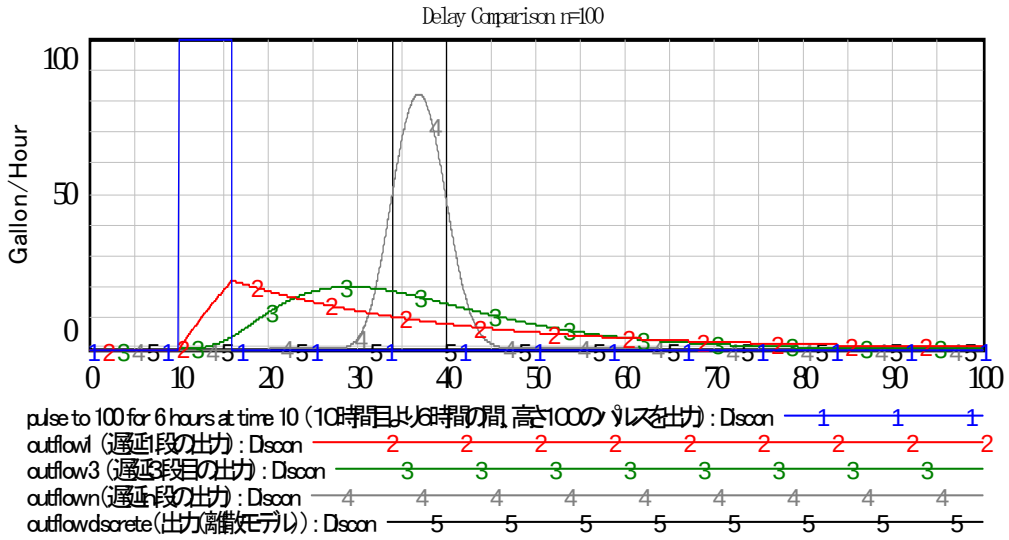


このモデルで、フロー1は、1stオーダーの遅れのフローです。フロー3aは3rdオーダー遅れ、フローnはn階オーダー遅れ(nはモデル定数)です。遅れのオーダーは遅れの中に含まれるストックの数で決まります。1stオーダーや3rdオーダーではストック数は明らかです。n階オーダーや離散時間系遅れはストックの数はストックの数を計算する方程式の内部構造の中に隠されてしまっています。

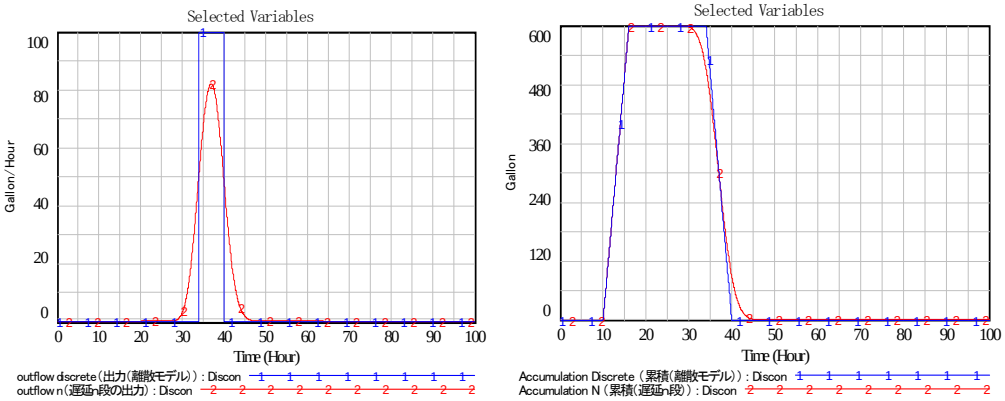
nが12の場合、モデルの振る舞いは次の様になります。



nが100のときは次の様になります。



大きなn値でシミュレートすると時間がかかりますが、このモデルで実験することは大変有用ですので是非やってみてください。nを増加させるにつれて、離散時間系の遅れに近くなって行きますが、実際には同じにはならないということが重要ですので観察してみてください。100階のオーダー遅れでさえ、離散時間系の遅れとの間にはフロー（出力）に関しては顕著な違いがあります。しかしながら、蓄積Nと蓄積（離散時間系）を比較すると、それほど違いはありません。



離散時間系の遅ではストック数は実際には無限ではないことに注意する必要があります。実際には、ストック数はProcessing time/TIME STEPになります。DELAY FIXEDおよびDELAY Nの様な高いオーダーの遅れ関数では、出力はTIME STEPの時点の値で一定に保持される様になっています。これは、離散時間系の遅れを連続時間系と混合して使っても構わないということを意味します。

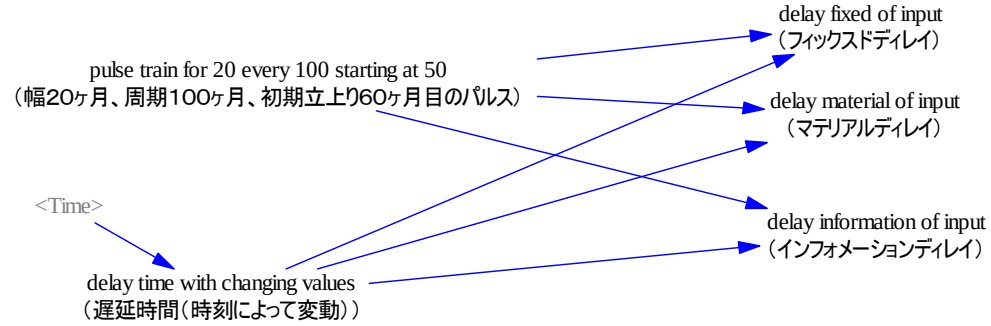
9.3 物質の遅れと情報の遅れ

離散時間系の遅れには基本的に3つのタイプがあります。DELAY FIXED、DELAY MATERIALおよびDELAY INFORMATIONです。遅延時間が一定の場合は、この3つの関数はすべて同じ様に作用します。遅延時間が時間とともに変わるときだけこれらに違いが生じます。

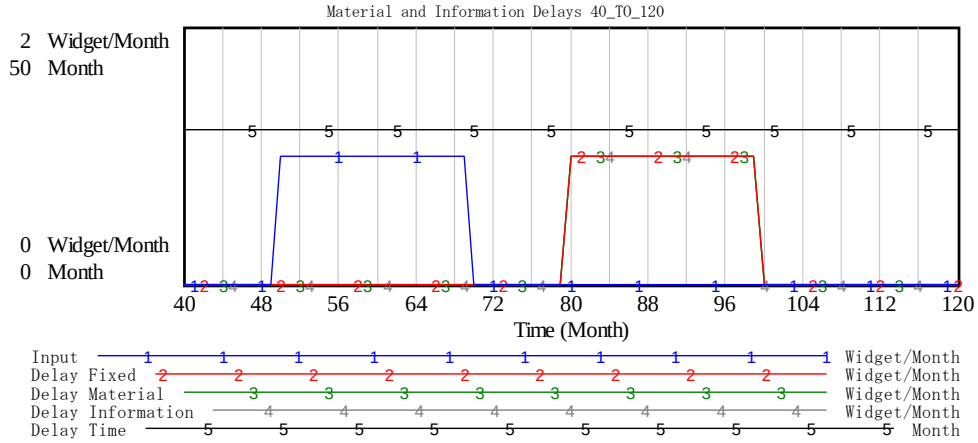
物質の遅れには、DELAY MATERIAL、DELAY1、DELAY3およびDELAY Nがありますが、投入された物質の量が保存されるという性質を持っています。すなわち、入力の蓄積は出力の蓄積と一致するということです。遅延時間が減少する場合、単位時間あたりの出力の量は増加するということを意味しますし、その逆も正しいです。

情報の遅れは、DELAY INFORMATION、SMOOTH、SMOOTH3およびSMOOTH Nがあります。情報の遅れでは入力の範囲を保存する様になっています。すなわち、遅延時間がいかに変化しようとも、関数は最も大きな入力値より大きい値を返したり、最も小さな入力値より小さい値を返したりしない様になっています。これは、情報の信号(例えば価格から目標生産に繋げる情報信号)に遅れが生じた場合を表現するために適切な仕様です。もし、物質の遅れの仕様を情報の遅れに対して適用したならば、仮に情報伝達速度が上昇したならば、実際には価格が一定だったとしても、増加した価格が生産セクターに伝達されてしまいます。このように、情報の遅れでは、伝達時間のいかんに関わらず、出力が増加・減少してはいけないのです。

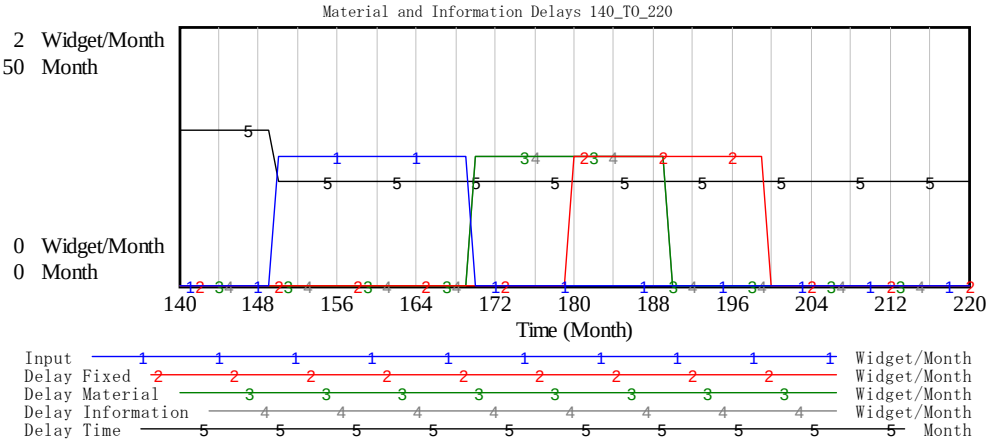
モデルdelay1はこれらの3つの機能、および遅延時間の変更に応じて、それらがどの様に変わるかを実証します。



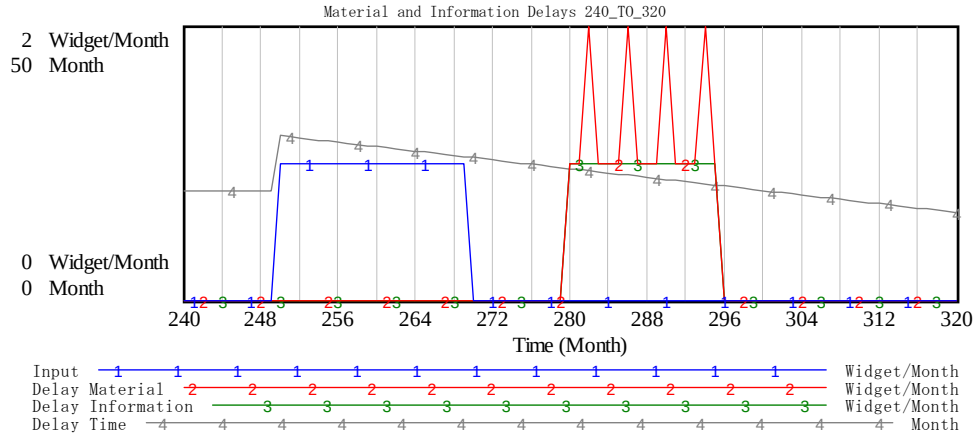
このモデルでは、遅延時間はこれらの違いを強調するためにセット・アップされています。時間50～100の間は、遅延時間は一定です。3つの関数の機能はすべて同じ結果を示しています。



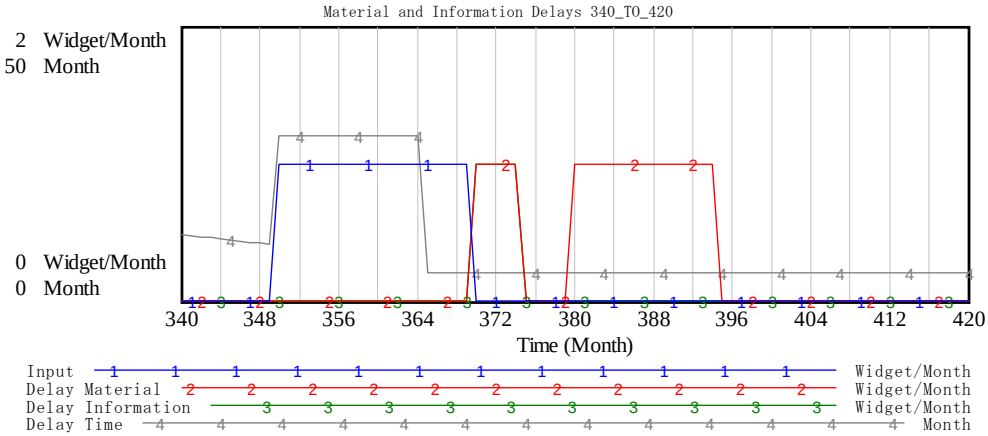
時間150～200では、遅延時間は先ほどとは異なる値で一定です。



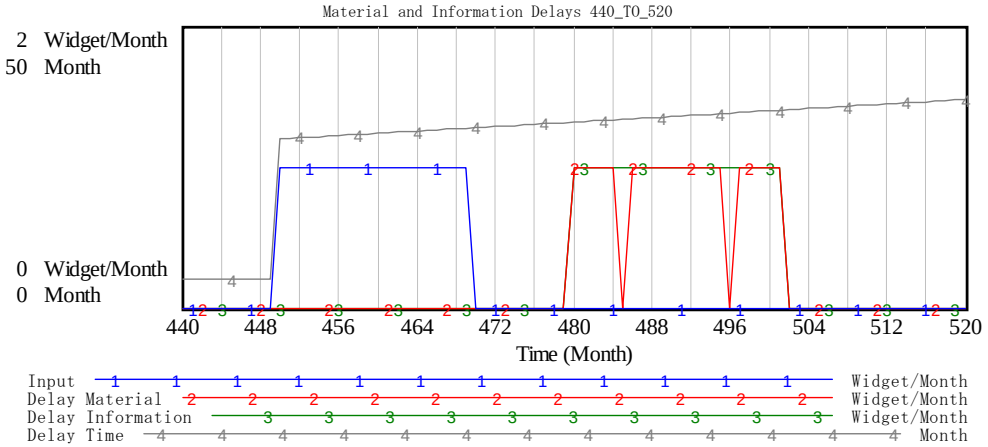
DELAY FIXED機能は、関数に最初(時間0)の時点で与えられた遅延時間(30)で処理します。他の2つの関数は、変更された遅延時間(20)を使用しています。遅延時間がどの様になろうとも、DELAY FIXED関数は常にその初期値を使用します。グラフを見やすくするためにDELAY FIXEDは次の例からは除外します。遅延時間が、時間とともに減少する場合は、次の様になります。



情報の遅れでは遅延時間が徐所に小さくなることによって、アウトプットの持続時間が短くなりますが、出力の値は一定です。これは、遅延時間が変化しても情報の中味は変化してはいけないということに基づいて仕様化されているからです。物質の遅れは遅延時間が徐所に小さくなることによって、同じ様にアウトプットの持続時間を短くなりますが、総出力が総投入量(20 Widgets)と同じになる様に、より多くの物質をアウトフローするようにアウトプットします。遅延時間がより急峻に減少した場合は意外な変化が起こります。実際には次の様になります。：



物質の遅れでは、先の投入されたものが処理されてアウトプットされるよりも、後で投入されて短い遅れ時間で処理されたものが、追い越して先にアウトプットとして現れます。他方、情報遅れでは、途中で情報の追い越しが起こる様な状況では、その様な情報は古くて利用価値がなくなっているものとみなされ、情報全体が廃棄されたものとして扱われます。最後に、遅延時間が増加している場合は次の様になります。



物質、情報の何れの場合も出力の期間が増加します。物質の遅れでは、何も出力されない時間が２回発生します。その一方で情報遅れは単に1Wigets/月で一定の状態を保っていることを意味します。

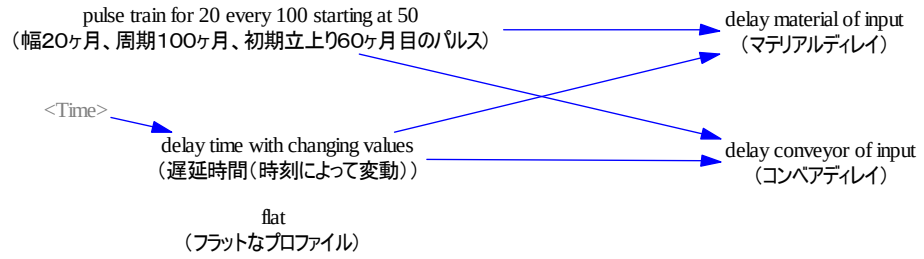
なお、物質の遅れに関して、後で投入したものが先に投入した物質を追い越して出力される様なことを避けるためには、DELAY CONVEYOR関数を使用するのが良いでしょう。

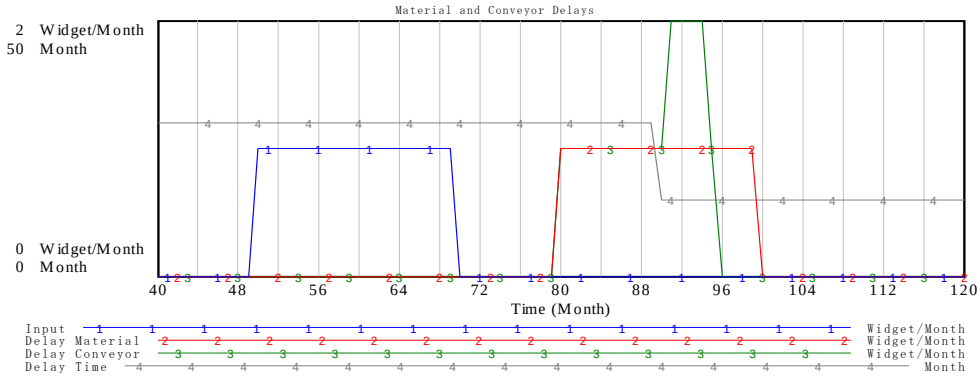
9.4コンベアー

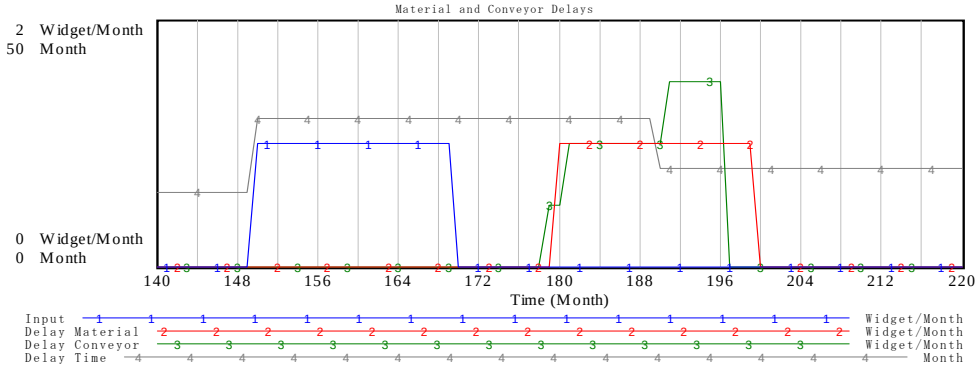
DELAY CONVEYOR関数では、コンベアーに材料を置き、コンベアーのスピードを上げたり下げたりする場合の遅れプロセスをモデル化することができます。DELAY CONVEYOR関数は、コンベアーというたとえは相応しくない様に感じるかもしれませんが、コンベアー上の材料が漏れることをモデル化できるようになっています。実はこの機能は大変役に立つのです。

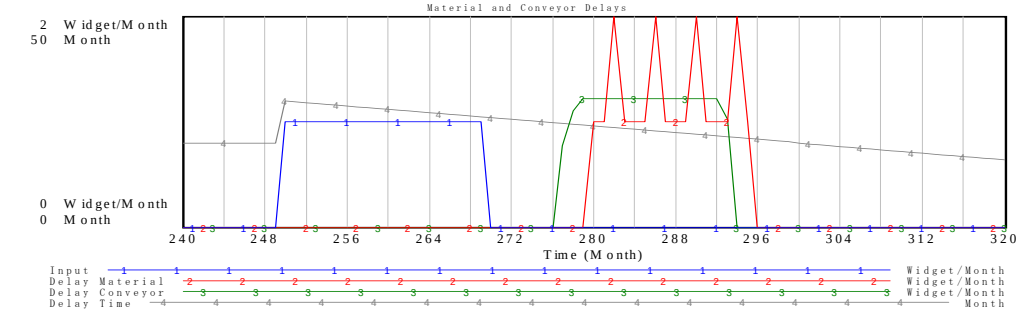
DELAY CONVEYOR関数が固定の遅延時間でかつ漏れがない場合には、そればDELAY MATERIAL(あるいはDELAY FIXED)として機能します。スタート時におけるプロセス中の物質の正確な分布状態を指定したいときにもコンベアー関数は役に立ちます。

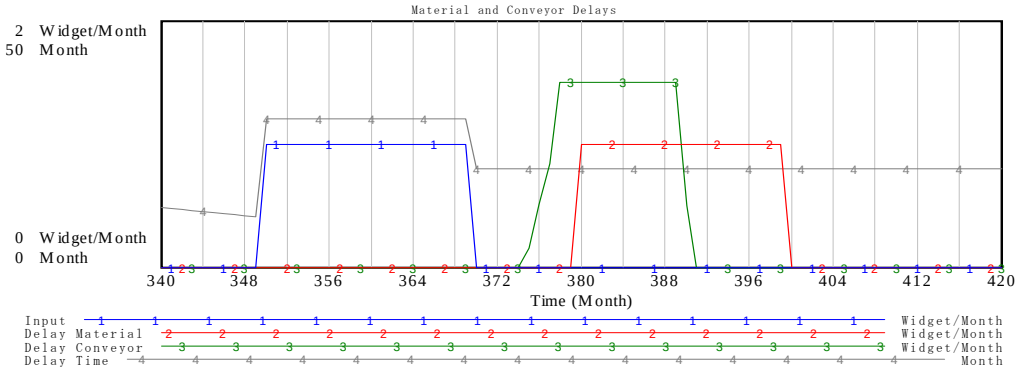
遅延時間が変更する場合、DELAY CONVEYORの振る舞いはDELAY MATERIALよりもはるかにより連続的に変化します。モデルdelay2.mdlは遅延時間を様々に変えてみて、DELAY CONVEYORとDELAY MATERIALの2つの機能を比較してみます。

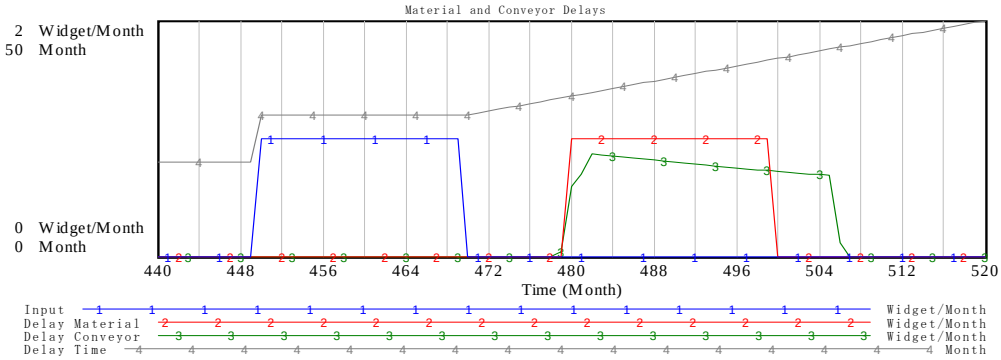






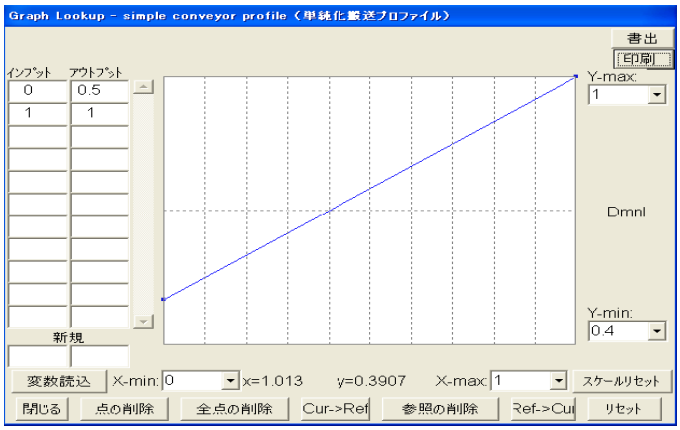




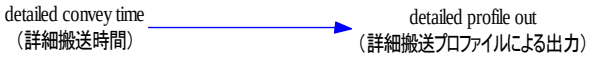


9.4.1 コンベアーの初期化

コンベアーを初期化する際に、コンベアー中の材料のプロファイルを指定します。このプロファイルは、コンベアーが出力する材料の量を時間軸にそって記述することで初期の分布を指示します。指定した材料の合計と初期の運搬時間にあわせて、Vensimは自動的にプロファイルの領域と範囲の尺度を合わせます。モデルconvey1.mdlはこのことをデモしています。例えば表関数を使用すると次の様になります。

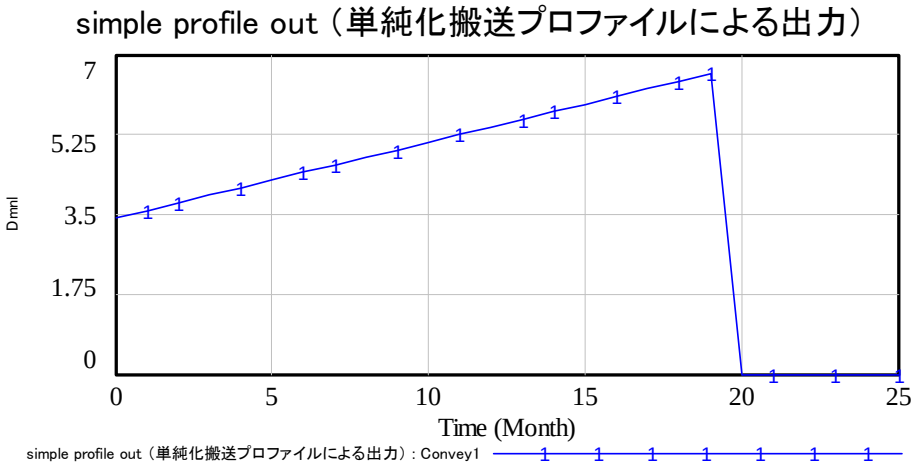


simple conveyor profile
（単純化搬送プロファイル）

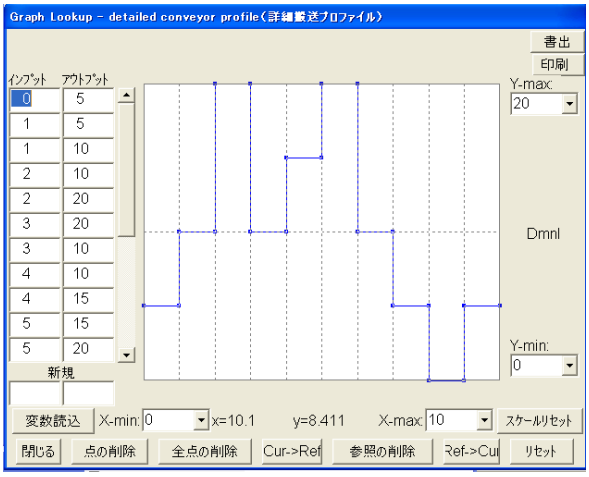


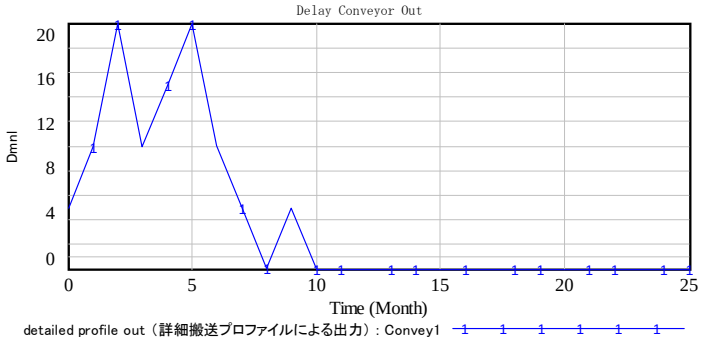
detailed conveyor
profile（詳細搬送プロファイル）

DELAY CONVEYOR関数を初期化することでこれらの出力を得ることができます。（入力は0です。最初に投入した材料だけを出力しています。）



材料の総量とそれが出力される時間は引数によって決定し、出力の形状を表関数で与えるようになっています。これは似たプロファイルを設定するためには便利です。そして、通常これで大抵はOKです。どの様な時期にどの様に出力するのかということを正確に指定するのであれば、より入念な表関数が必要になります。例えば次の様な表関数です。





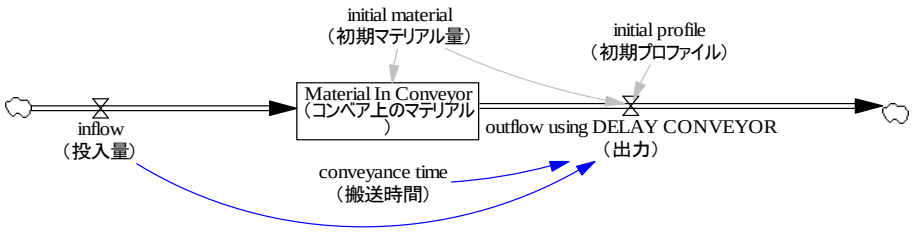
最初に在庫が100あったとしましょう。遅延時間が10のとき、detailed profile outとして次のように出力します。

Time	0	1	2	3	4	5	6	7	8	9	10
detailed profile out	5	10	20	10	15	20	10	5	5.00E-06	5	0

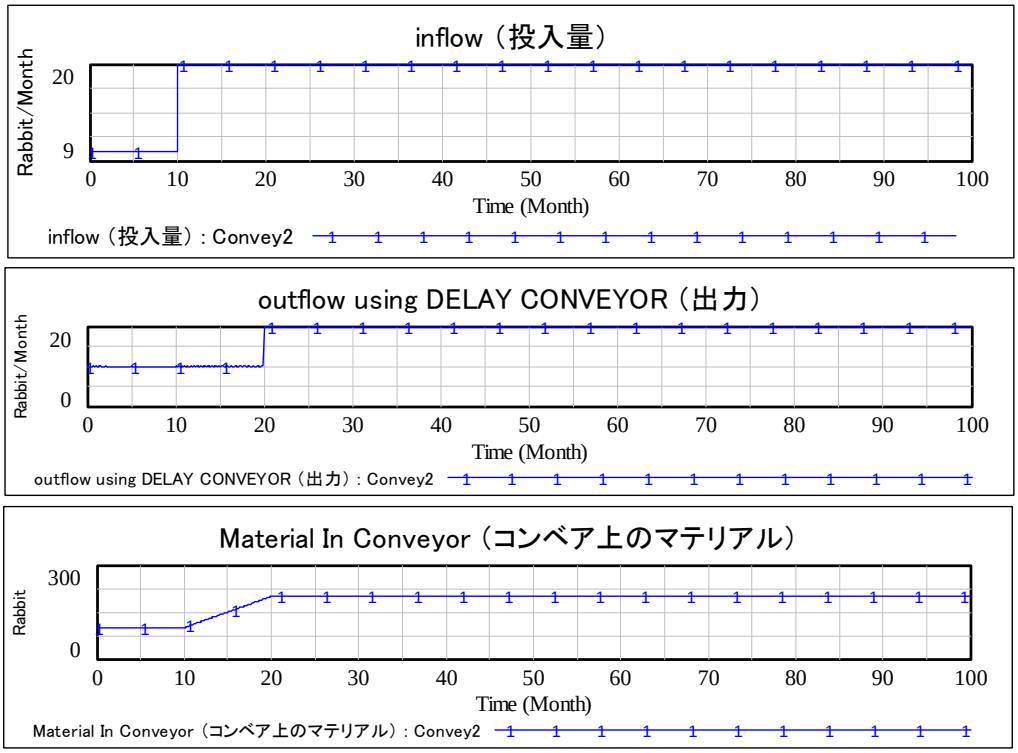
今回あなたが変更したのはストックの合計値と(こちらがより重要なのですが)遅延時間でした。これらのきっちりの数字はあまり長く続きませんでした。時間8に発生する小さな丸め誤差にも注目してください。DELAY_CONVEYORを使用する場合は、必ずしも整数値の結果となるとは限りません。

9.4.2 コンベアー中の材料

漏れのないコンベアーは、コンベアー中の材料の量は次に示す構造で決定できます。(convey2.mdl)

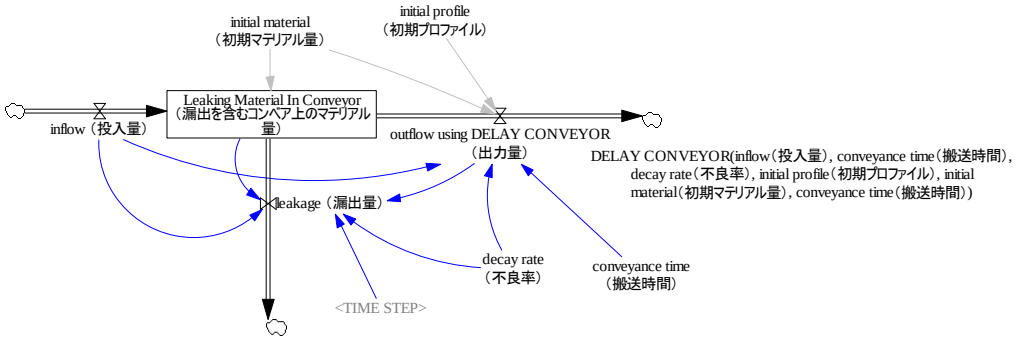


ここで初期化式は分かりやすくするために明記しています。



コンベアーから材料の漏出がある場合は多少構造を変える必要があります。漏出は、現在コンベアー中にある材料だけでなく、これから入ってくる材料についても起こります。別な言い方をすれば、もし時間0に何かを投入したとすれば、時間1までにいくらかの漏れが起こるということです。

これらを表現すために構造を改善すると次のとおりになります。(convey3.mdl)



また、漏出は次のように定式化できます。

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

この方程式の中央の項は、これから入ってくる材料に関しても漏出が起こるために、前もってインフローとアウトフローの差を計算に加えておくためのものです。もし無視したとしても比較的小さな誤差となります。そしてTIME STEPは小さいので実際の影響は更に小

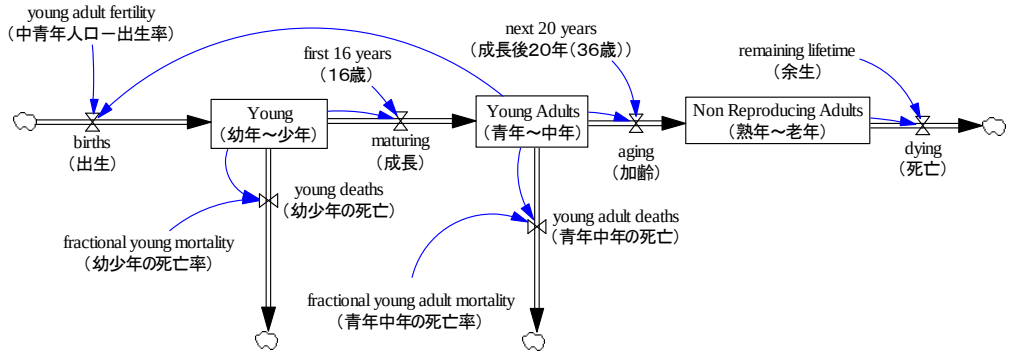
さくなります。単純にこれを無視してしまい、単純に定式化することは多くの場合賢明なことです。

leakage= Leaking Material In Conveyor * decay rate

この様に定式化しても、大抵の場合は、大きな誤差を引き起こすことはないでしょう。このように単純化したことによる誤差は、(漏出があれば、アウトフローの累積はインフローの累積より少なくなるので)正の方向に蓄積して行く傾向があります。

9.4.3 コンベアーを人口動態に適用した例

離散時間系の論理が上手く適用できるものに人口動態学があります。そして、標準的なアプローチとしてコホート(Cohorts)は良く用いられます。(convey4a.mdl)

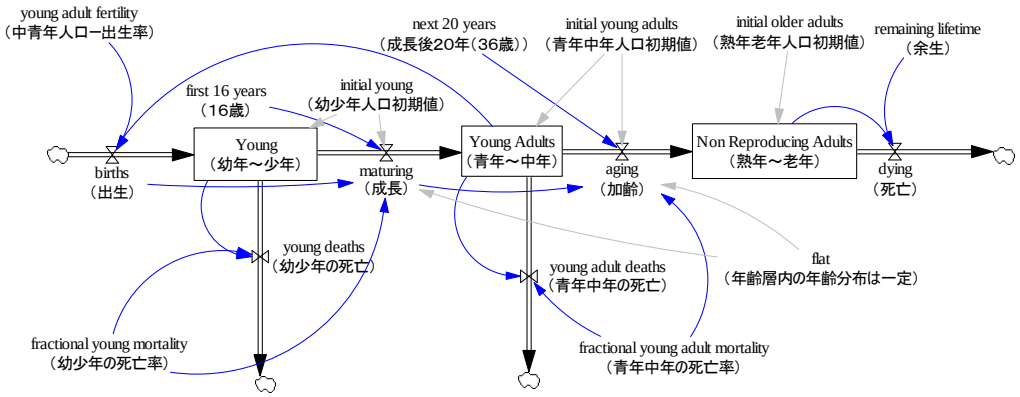


卯

上記のモデルを定式化すると次の様になります。

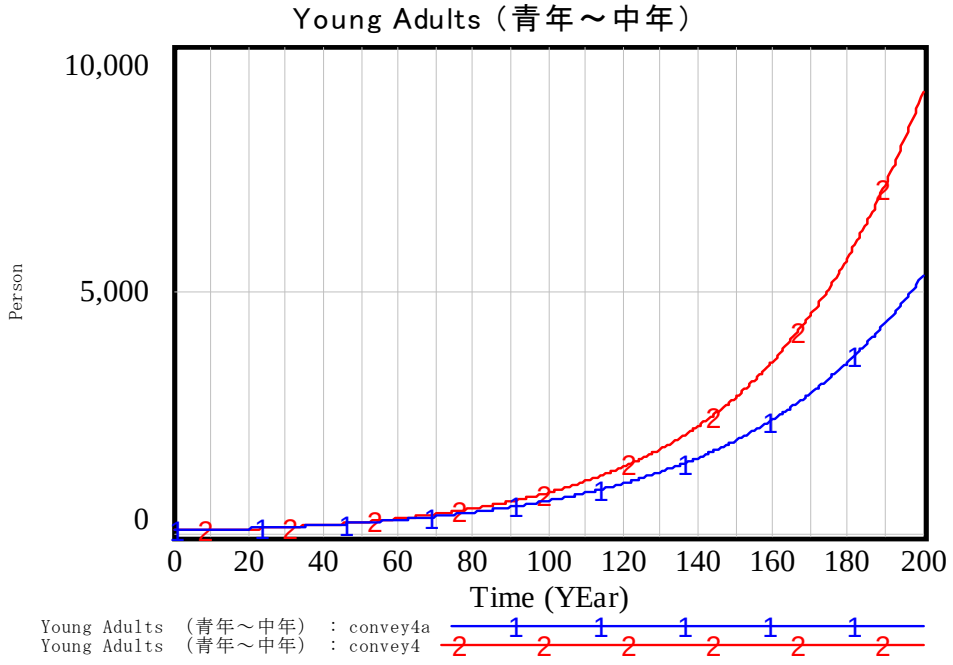
Maturing = young/first 16 years

指数関数的崩壊が持つ性質のために、時間0にbirthsが増加すれば、1年後にYoung Adultsの増加を引き起こしてしまいます。(成長に要する時間は決まっているのでこんなことはありません。)このような矛盾を、コンベアー関数を使って解決するために定式化をやり直します。(convey4.mdl)。



maturingとagingをDELAY CONVEYORを使用して計算するという点を除いて、convey4aとほとんど同じです。しかしながら、その振る舞いはパラメータを同じ様にセットしても、著

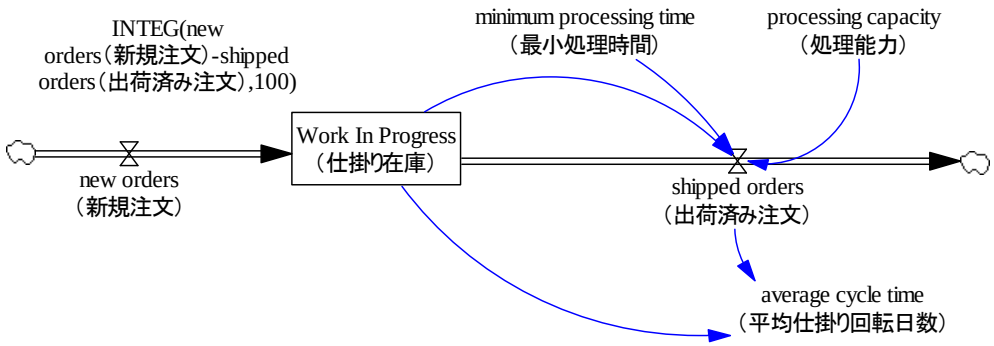
しく異なります。


Young Adults（青年～中年）
10,000
5,000
0
Person
0
20
40
60
80
100
120
140
160
180
200
Time (YEar)
Young Adults （青年～中年） : convey4a
Young Adults （青年～中年） : convey4

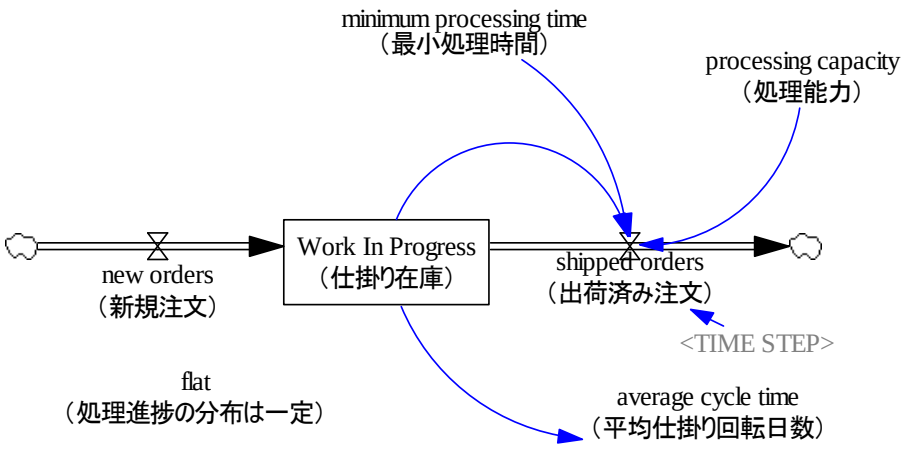
9.5 待ち行列

待ち行列とは、物事を置き次にそれを取り去るということが行われる単純な場です。待ち行列の最も一般的な例は、何かを待つために列をなしている行列です。また、その様な待ち行列は、ファーストイン・ファーストアウト(最初に到着した人が最初にサービスを受けること)を意味するFIFOキューと呼ばれます。VensimにはFIFOキューによる処理を簡単に行うことができる関数がいくつかあります。

Vensimの中のQUEUE関数にはすべて2つの部分からなっています。1つは、特別なストックです。物事を蓄積できるという意味では普通のストックと同じです。普通のストックは物事をまとめて塊にして一つの数字にしてしまいますが、この特別なストックは、いつ物事が到着したのか、あるいは出たのかといった物事の軌跡を保持する機能を持っているという意味で特別です。例えば注文処理について考えていると仮定してください:
(queue1.mdl)



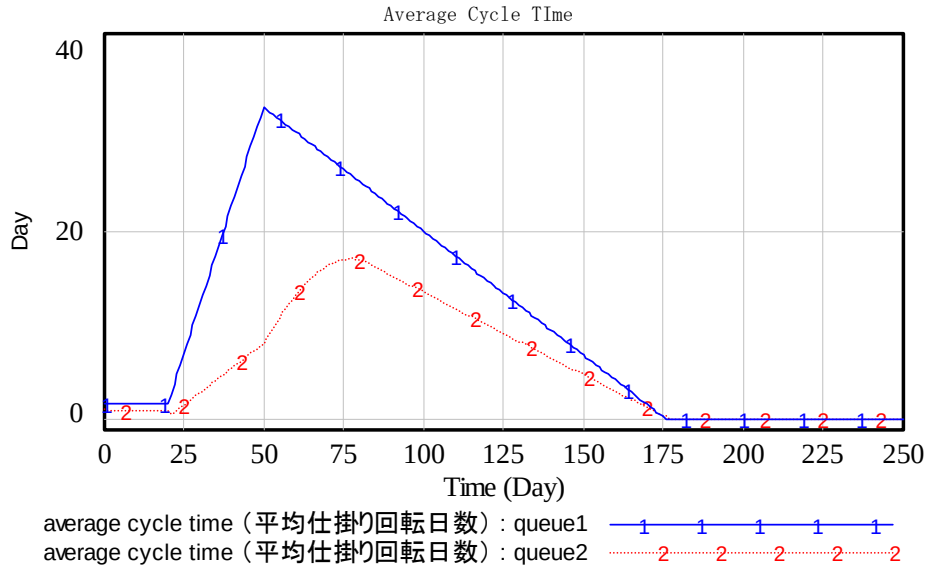
上記は、代表的な従来の連続時間系によるアプローチです。このアプローチでは待ち行列は利用しません。
Average cycle timeは次の様に計算できます。
average cycle time=ZIDZ(Work In Progress,shipped orders)
待ち行列を使用して定式化して比較してみましょう。



図形は、flatという表関数を追加したことを除いてほとんど同じです。そして、shipped ordersからaverage cycle timeへ向かう矢印が無くなっています。変更された方程式は次のとおりです:

```
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
```

これら2つのモデルのシミュレーションを比較すれば、averate cycle timeを除けば、ほとんどの変数は同じ様に振る舞っていることが分かるでしょう。



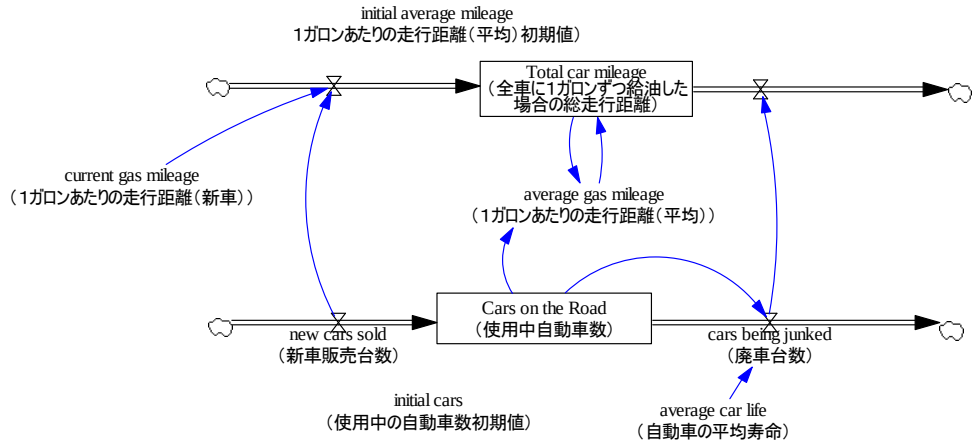
連続的な定式化では、average cycle timeは30日間で2から32まで増加しています。それまでは何も生産しておらず、すべての注文をTime 20に受けたのであれば、そのようになるに違いありません。QUEUEを使用して定式化すれば、より正確な答えを示すことができます。

一般に、QUEUE関数の価値は、正確なサイクルタイムを示すことができ、性能を計る尺度にすることができるということにあります。

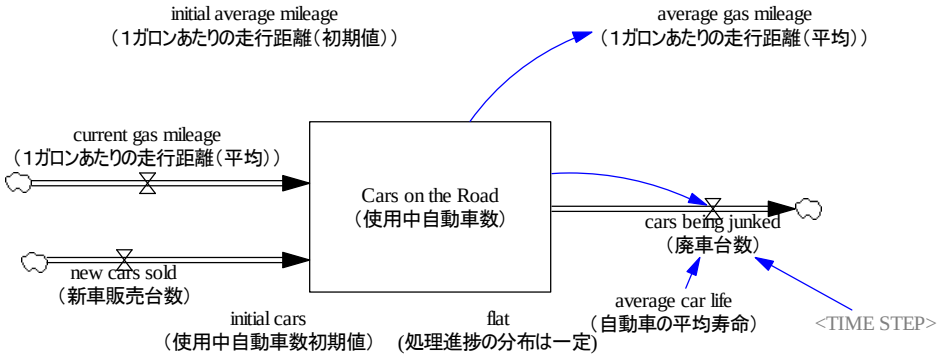
9.5.1 属性を備えた待ち行列

待ち行列の中でどれくらいの時間待ったかをトレースすることに加えて、待ち行列の中で、ある属性がどの様に分布しているかということを知ることは多くの場合有用です。離散時間系のイベントシミュレーションでは、個々の実体に着目しシステム内でどのように機能したのかを追跡するというアプローチをとります。下添え字を使用して詳細に行うとも意味があることも時々はありますが、それほど詳述でなくとも良い場合もあります。属性を備えた待ち行列を使えば、下添え字を使ったフル装備ではなく、より簡便に詳細を知ることができます。

例として、様々な製造年度の自動車の燃費をトレースすることを考えてみます。この問題に対する連続時間系における典型的なアプローチはここに示す様なコ・フローを使用することでしょう。(queue3.mdl)



コ・フローでは、単位Car*Mile/Gallonを持った分離したストックを用いて属性を追いかけることができます。コ・フローはFIFOアプローチよりはむしろ、負の係数を持った指数関数による滞留分布に従って作動します。FIFOを使用して定式化してモデルにすると次の様になります。(queue4.mdl)



抽象的なストックであるtotal car mileageを削除して、Cars on the Roadに対して2つのインフローを定義しています。このうちの1つが属性のフローです。しかし、これは大変分かりやすく表現している様なものです。実際のCars on the Roadの式は次のとおりです。

```
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)
```

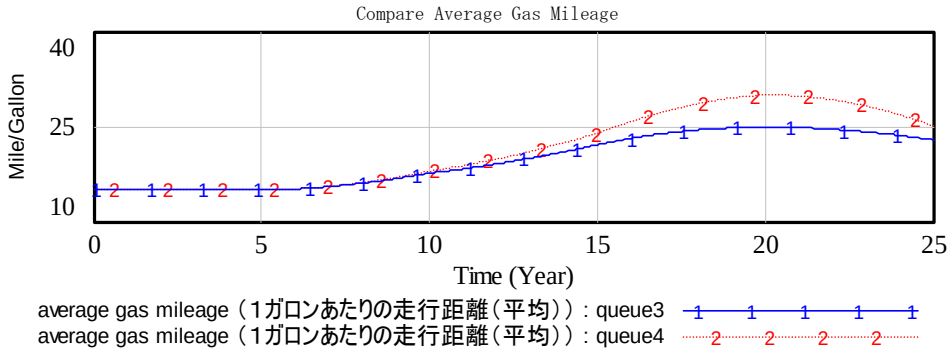
この定式化は実際には2つのレベルの方程式を立てており、それらを同時に計算しています。Average gas mileageは次の方程式を使って計算します。

```
average gas mileage= QUEUE ATTRIB AVERAGE(Cars on the Road, -1)
```

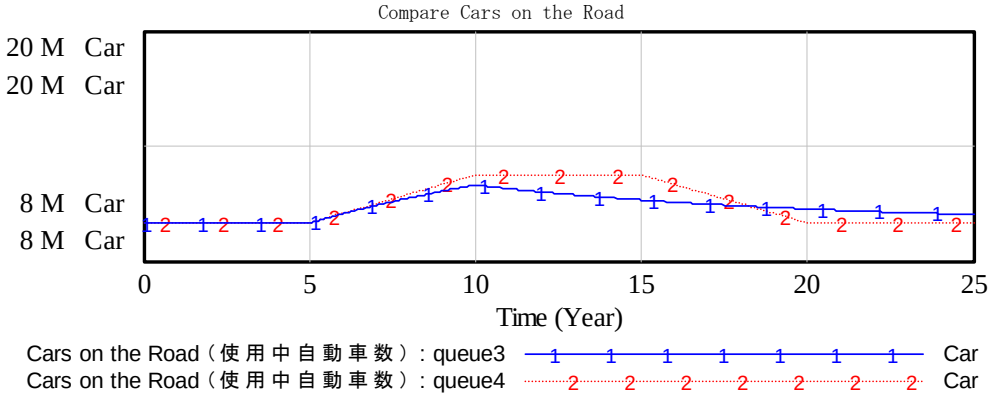
最後に、cars being junkedは次の様に置き換えることができます。

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

avegage gas mileageを見ると2つのモデル間に重要な違いがあります:



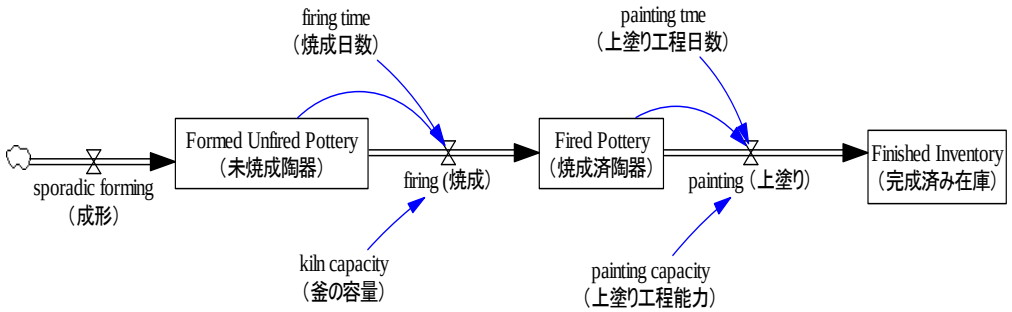
勿論、Cars on the Roadにも違いがあります。



しかしながら、サイクルタイム計算と異なり、これらのうちの1つが明らかに間違いでもう1つが明らかに正しいという話ではありません。両方とも、近似に過ぎないという意味で同じです。何故ならば、自動車は負の係数を持つ指数滞在時間分布に従って廃車される訳ではありませんし、FIFOキューによるプロセスで廃車される訳でもないからです。

9.6バッチ遅れ

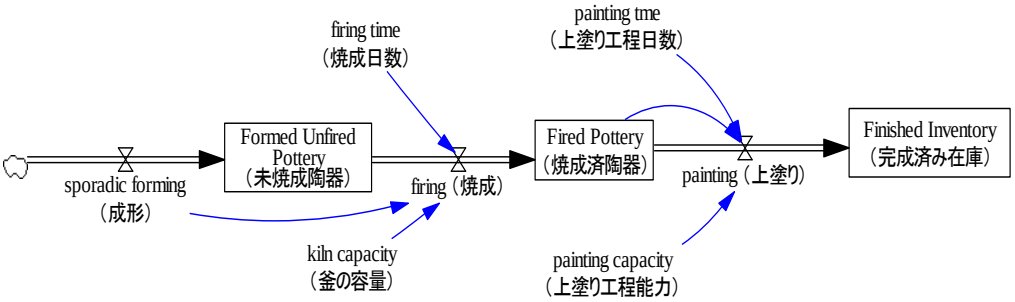
ほとんどの遅れは、何かを取り込み、平準化して出力します。(9.1項で見た様に、遅れが高次になればなるほど出力が滑らになるとは限りません。) ある場合には、インフローを滑らかにするのではなく、滑らかなインフローをバッチ (一括処理) として分割する必要がある場合も起こります。この様な場合には、DELAY BATCH関数を用います。例えば、大きな窯による陶磁器の生産を考えてみてください。連続時間系で表現をしたならば、典型的には次の様な構造になるでしょう。(batch1.mdl)



ここで、firingは次の式によって与えられます:

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

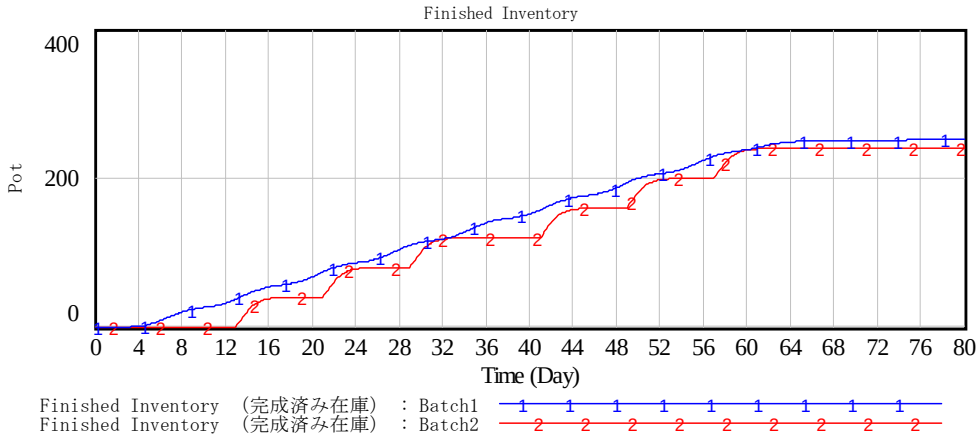
しかしながら、窯に陶磁器を入れるということを想定するならば、バッチプロセスでモデルを作成する方がより正確です。(batch2.mdl)



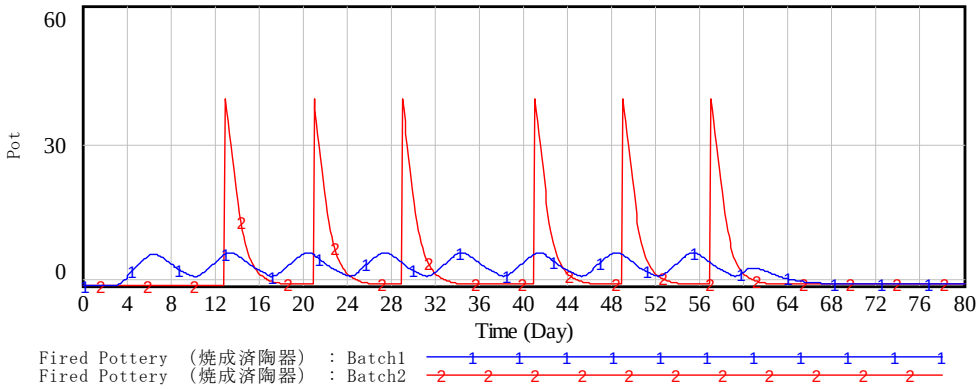
その場合は、fireは次の式で定式化されます。

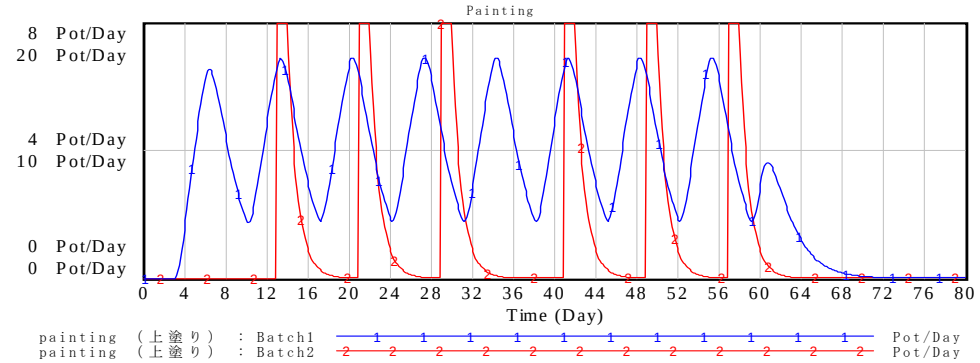
firing= DELAY BATCH(sporadic forming, kiln capacity, firing time, 0, 0, 0)

バッチプロセスによってモデル化した場合の最終産出物は、連続時間系による場合に比較すると「時々発生する」という特徴を持ちますが、それ以外は多くは変わりません。



しかし、窯から出力される焼成された陶磁器の量を見ると、連続時間系でモデル化した場合とバッチプロセスでモデル化した場合では大きく結果が異なってきます。その結果として１日あたりの絵付け工程の処理量は大きく異なってきます。





窯から焼成された陶磁器が出てくる様に、ある活動が「時々起こる」というタイプであり、その「時々起こること」が今回の上塗り工程の様にキャパの活用が制限する傾向がある場合、DELAY BATCHを使うことによって、より正確な結果を得ることができます。

以上