Tutorial – Simple Nonlinear Model-Based Process Control

R. Russell Rhinehart

Keywords: nonlinear, model-based, process control, simple

Table of Contents

1. Introduction

Often, nonlinearity is the primary problem for single-input-single-output (SISO) chemical process control. One solution is to design process for control success – design the process for linear responses, or for large inventories to reduce interaction and to temper upsets. However, design is an act of balancing multiple objectives; and other desirable issues such as capital cost, flexibility, resource use, energy integration, and sustainability are usually sacrificed to accommodate process control. Practicable techniques for nonlinear control can ease design constraints and permit the operation of more competitive processes.

When there are nonlinear processes, gain scheduling is a conventional control-oriented solution to accommodate nonlinearity. In gain scheduling the Proportional-Integral-Derivative (PID) controller coefficients are changed to reflect the operating region. In gain scheduling the engineer uses process knowledge to create tuning values, which are either placed in a look-up table or expressed in equations. The controller first looks-up (or calculates) PI (or maybe PID) coefficient values, then it uses the conventional PI(D) algorithm for control.

However, if the engineer's process knowledge is expressed as either a dynamic or steady-state nonlinear model, it is about as easy to implement a process-model based controller (PMBC) as gain scheduling. Several process-model based control approaches have found success in industrial applications: There are several commercial products for relatively simple nonlinear process-model based controllers, engineers have implemented versions in-house, and control integrators and service providers have been implementing model based controllers for decades.

SISO or MISO (multi-input single-output) process-model based control (PMBC) has several advantages over either PI(D) or gain scheduled PI(D). PMBC has a single tuning parameter, nonlinear compensation throughout the entire operating range, preservation of process knowledge, and continuous monitoring of the process – for health, predictive maintenance, and constraint recognition. For multi-input-multi-output (MIMO) processes, PMBC can decouple nonlinear interaction, balance deviations from set points when manipulated variable (MV) constraints are hit, and determine economic optimum MV values when there are extra degrees of freedom (DoF).

In contrast to the "large model", multi-step-ahead, advanced process control (APC) techniques characterized by horizon predictive controller (HPC) or model predictive control (MPC), the one-step-ahead controllers presented here can solve many problems and can be implemented by a process engineer.

This tutorial will present the concepts behind a MISO nonlinear process-model based controller that can be implemented in-house. Process-Model Based Control (PMBC) uses a simple dynamic model of the process, a single tuning parameter, tracking of process characteristics for process condition monitoring, and no integral windup.

Finally, this tutorial will show how to implement SISO and MIMO PMBC and handle the cases of excess and inadequate DoF.

Why use simple process-model based control?

1. It preserves and transmits process knowledge, as contrasted to FOPDT approximations.
2. Once tuned, it remains tuned throughout the operating range
3. There is only one aggressiveness factor, not several
4. When incrementally adjusted, the model parameter value tracks the process behavior for process monitoring of process health
5. The model forecasts constraints which can be useful in higher level optimization
6. The same model can be used in other applications – optimization, training design

2. Process Models

Process-model based control starts with the engineer's process model. Such models are often developed from first-principles (elementary phenomenological or mechanistic models, not full rigor), but models may also contain empirical features, or they may be purely empirical. Modeling may start with either a steady-state or dynamic model.

The following examples illustrate first-principles process modeling for control. They are taken from a variety of common processes to help reveal widespread applicability. They are also chosen to reveal

nonlinearity, nonstationary, disturbance, and interaction attributes that can be handled by process-model based control.

Model Example 1: Flow Rate Response to Controller Output.

If a valve is installed in a process line, undergraduate fluid dynamics or process control texts use pressure drop through pipes, valves, and fittings to derive the steady-state incompressible fluid flow rate response to valve stem position:

$$\dot{Q} = \sqrt{\frac{\Delta P}{a + G / Cv^2 \,/\, f^2(x)}} \tag{1}$$

Where $\dot{Q}$ is the volumetric flow rate, *ΔP* is the static pressure driving force over the entire pipe section, *a* represents the piping system friction loss coefficients, *G* is the fluid specific gravity, *Cv* is the valve coefficient of discharge, *x* is the valve stem position, and *f(x)* is the inherent valve characteristic.

Characteristically, the dynamics of fluid acceleration in response to a valve stem position are much faster than the valve position response to the actuator (which responds to air pressure with a time-constant of about ¼ to 2 seconds). Accordingly, the dynamics that dominate the fluid flow rate response to the controller output is often that from valve stem position, and it is reasonable to describe valve stem position as a first-order response to controller output.

$$\tau \frac{dx}{dt} + x = 0.01u \tag{2}$$

Where τ is the time-constant for the valve actuator, and *u* is the controller output in %. The 0.01 value converts % into valve stem position as a fraction.

This set of equations represents linear dynamics of Equation (2) followed by a nonlinear characterization of Equation (1), and is often termed a Weiner model. Coefficients in Equation (1) can be evaluated by engineering analysis of the process. The time-constant for Equation (2) can be revealed by the observed flow rate transient response to a simple controller output step test.

Model Example 2: Mixing in a Continuous Flow-Through Tank

Consider two streams of concentrations c_1 and c_2 entering a tank at a stationary liquid level. Perhaps F_1 and c_1 represent wild flow properties and F_2 is the controlled flow. The net inflow concentration is

$$c_{in} = \frac{c_1 F_1 + c_2 F_2}{F_1 + F_2} \tag{3}$$

Note that c_{in}, the net inflow concentration, is a nonlinear response to F_1. Usually, the mixing dynamics in the tank defines the outflow concentration as

$$\frac{V}{F}\frac{dc}{dt}+c=c_{in} \tag{4}$$

Where *V* is the tank volume and *F* represents the total flow rate (=F_1+ F_2), the ratio *V/F* has the units of a time-constant.

This represents a static nonlinearity of Equation (3) followed by linear dynamics of Equation (4), which is often termed a Hammerstein model. Equations (3) and (4) can be combined to represent one nonlinear model:

$$\tau\frac{dc}{dt}+c=\frac{c_1F_1+c_2F_2}{F_1+F_2} \tag{5}$$

While developed here as separate equations, then combined, the same result, Equation (5), could be derived from a transient material component balance on the tank.

Model Example 3: Heat Exchange

Consider that steam is used to heat a process fluid and that the dynamics of the process T response to steam flow rate can be modeled as first-order. An elementary steady-state energy balance reveals that the outlet T response will be:

$$T_{out}=T_{in}+\frac{\lambda\dot{Q}_s}{F\rho Cp} \tag{6}$$

Where T_{in} and T_{out} represent the inlet and outlet temperatures of a process fluid at flow rate *F* with density and specific heats of ρ and *Cp*. The steam flow rate is $\dot{Q}_s$ with latent heat of vaporization λ.

Assuming that valve dynamics are negligible (x=u/100 in a relatively short time) and that the installed valve characteristic is f(x)=f(u/100), the steam flow rate response to controller output is:

$$\dot{Q}_s=Cvf(u/100)\sqrt{\Delta P/G} \tag{7}$$

Combining with first-order process dynamics produces the dynamic model:

$$\tau\frac{dT}{dt}+T=T_{in}+\frac{\lambda}{F\rho Cp}Cvf(u/100)\sqrt{\Delta P/G} \tag{8}$$

Model Example 4: MIMO In-Line Hot and Cold Fluid Mixing

Consider that hot and cold fluids are mixed in-line (like a home shower), and that fluid flow rates F_h and F_c are manipulated by valves with a linear-ish installed characteristic. Then their steady-state flow rate response to signals u_1 and u_2 are

$$F_h = k_h u_1 \tag{9}$$

$$F_c = k_c u_2 \tag{10}$$

The values for k_h and k_c could be empirically determined.

From energy and material balances, the steady-state mixed temperature and total flow rate are:

$$T_{in} = \frac{T_h F_h + T_c F_c}{F_h + F_c} \tag{11}$$

$$F_{in} = F_h + F_c \tag{12}$$

Assuming first-order dynamic responses of temperature and flow rate, the net models for the flow rate and temperature dynamic response to u_1 and u_2 are:

$$\tau_T \frac{dT}{dt} + T = \frac{T_h k_h u_1 + T_c k_c u_2}{k_h u_1 + k_c u_2} \tag{13}$$

$$\tau_F \frac{dF}{dt} + F = k_h u_1 + k_c u_2 \tag{14}$$

Model Example 5: Car Speed

Start with Newton's Law of Motion for linear translation of the car, and express acceleration as the rate of change of velocity. This explicitly provides the evidence of the time dependence in the state variable, the controlled variable, velocity.

$$F = \frac{ma}{g_c} = \frac{m}{g_c}\frac{dv}{dt} \tag{15}$$

Here, *m* is the car mass, *F* is the accelerating force, and g_c is the dimensional unifier (32.17 lbf-s^2/lbm-ft, or 1 N-s^2/kg-m).

Expand Force as having three elements as shown in Equation (16). One accelerating force (or a decelerating force) is from the engine. Resistance to motion, decelerating drag force, is from any number of aspects (wind resistance, rolling friction, etc.). Gravity helps accelerate when going downhill and decelerate when going uphill. In Equation (16) θ is the angle of climb from the horizontal and g is the gravitational acceleration (at sea level 32.17 ft/s^2, 9.807 m/s^2). For this simple model, consider that the torque from the engine is linearly converted to a force on the car, and that torque is proportional to

accelerator pedal position, u. Also, for this simple model, assume that drag forces are proportional to the square of velocity.

$$F = ku - av^2 - \sin(\theta)mg / g_c \tag{16}$$

Note: drag coefficient "a" changes in time as the car moves from a smooth road to a rough one or dry sand changes to wet sand. The drag coefficient also changes as the air density changes (pressure, humidity, temperature, moisture). And note that the velocity in the drag element is not the car land velocity, but the car air velocity which changes with wind speed and direction.

Combining Equations (15) and (16), and rearranging into the standard from in which the state variable appears to the first power generates the model.

$$\frac{m}{avg_c}\frac{dv}{dt} + v = \frac{k}{av}u - \frac{\sin(\theta)mg}{avg_c} \tag{17}$$

Here, it should be evident that there are several nonlinear and nonstationary aspects. Even though the accelerating force is a linear function of pedal position, u; the model indicates a nonlinear response of velocity to pedal position in which the gain, *k/av*, is not a constant. The model is nonstationary (changes in time) for several reasons. Coefficient "a" and angle, θ, both represent external disturbances that change in time, and velocity changes in time. Because the time-constant, m/avg_c, changes with both coefficient "a" and velocity, because the gain changes with coefficient "a", and because the gravitational effect changes with "a", v, and θ the model is nonstationary.

Generic Format for Model Equations

This paper will represent the generic dynamic model as

$$\frac{d\underline{y}}{dt} = \underline{f}(\underline{y}, \underline{u}, \underline{d}, \underline{p}) \tag{18}$$

Where y is the vector of process responses, u is vector of the process manipulated variable (MV) controller influences on the process, d represents measureable disturbances, and p represents model parameters. Process model-based control (PMBC) will be derived from the generic Equation (18).

It is relatively easy to rearrange Equations (5, 8, 13, 14, and 17) into the form of Equation (18) by isolating the derivative term on the left hand side of the equation. Then follow the directions in the next section to formulate the PMBC controller.

A Weiner model (first order dynamics leads a nonlinear response), such as that in Equations (1) and (2), can also be placed in the same generic structure. Consider this SISO Weiner model, in which the function symbol g() represents a steady-state (SS) relationship.

$$\tau\frac{dx}{dt} + x = ku \tag{19}$$

$$y = g(x, d) \tag{20}$$

Take the time derivative of Equation (20) using the chain rule

$$\frac{dy}{dt} = \frac{\partial y}{\partial x}\frac{dx}{dt} \tag{21}$$

and substitute *dx/dt* from Equation (19) and *x* from the inverse of Equation (20) into Equation (21)

$$\frac{dy}{dt} = \frac{\partial y}{\partial x}[ku - g^{-1}(y, d)]/\tau = f(y, u, d, p) \tag{22}$$

Equation (22) is the SISO version of Equation (18).

You will need to derive a dynamic model in the form of Equation (18) or (22) to implement PMBC.

Modeling Perspectives

These, and similar process models do not pretend to be rigorous; and fortunately for control, the best in model accuracy is not needed. Consider this: A model is not quite perfect in either dynamics or gain, and the PMBC decides a control action that is only 75% perfect, leaving 25% error. In simplistic qualitative reasoning, at the next control action the PMBC will correct the 25% error by 75%, leaving 25% of 25% uncorrected. With a rule of thumb that there should be 30 control actions within a process settling time, the continual "steering" by the controller will have the process at the set point, or close enough with an undetectable $0.25^{30} \approx 10^{-18}$ fraction remaining to be corrected. Drivers can steer their car through turns. If the first move of the steering wheel does not align the front tires perfectly to the road curve, continual readjustment will. Balancing precision with sufficiency, a model that would represent a homework grade of "B-" or "C+" is fully adequate for control.

Common evidence supporting this from standard control practice is that a PID controller works adequately over a reasonable range of process conditions. It may be a bit aggressive where either the process gain or deadtime are high, or a bit sluggish elsewhere. But, it can still regulate and track set points. APC, based on linear finite impulse response models (the dynamic matrix) is a linear controller. But it, too, works over a reasonable range of nonlinear process characteristics. The controller might be graded a perfect "A" in one operating region, but it still works in regions where its model is just a mediocre representation of the process.

This robustness to model imprecision has several advantages. It lessens the burden on the control engineer to achieve model perfection. And, it permits the controller to remain functional as a process continually changes because of fouling, reactivity, yield, filter/screen accumulation, or piping modifications.

Further, simple process models, such as those expressed in Equations (1) through (14), often represent desirable computational features of execution simplicity and low computational complexity and storage.

Additionally, such simple models often represent the engineer's understanding of the process, and reflect methods already in use for process analysis, device sizing, and analytical trouble-shooting. The process engineer can develop such models.

These models do not exactly match the process. Neither would a best effort at developing a rigorous model. There is uncertainty in coefficients representing, for example, heat transfer fouling, catalyst reactivity, tray efficiency, friction losses in pipes and fittings, concentration of non-key components, amount of non-condensable gases, etc. There are also simplifications such as presumed first-order dynamics, the Bernoulli Equation, the ideal gas law, etc. There is also uncertainty on the value of disturbance and influence measurements. Accordingly, when used for control the models need real-time correction.

3. Simple Process-Model Based Control – What it is and is not.

This tutorial presents several versions of process-model based control algorithms which can be implemented in-house and which have been developed to solve the problem of 1) nonlinear process response to manipulated and disturbance variables, 2) multi-variable interaction, and 3) MV constraint events. We will start with SISO model-based controllers. However, the controllers can integrate measured disturbances making them feedforward/ratio/MISO in one sense; but with only one MV, I'll refer to them as SISO.

The process-model based controllers described here are intermediate from PID to what is classified as ARC (Advanced Regulatory Control) and APC (Advanced Process Control) [1]. The controllers described here are not horizon-predictive controllers. They would not be classified as APC which includes products such as MPC, DMC, HPC, IdCom, Delta V Predictor, Perfector, etc. These and other horizon predictive controllers use a model to forecast what the model will do in the future, and optimize a forecast sequence of MV moves to best shape a future trajectory of the model subject to future constraint avoidance and/or to handle ill-behaved dynamics.

Although, the approaches described in this tutorial have models, and can be imbedded in horizon-predictive, constraint avoidance algorithms, techniques presented here are not the "big" models of APC products, nor do they have horizon predictions. The approaches here are each based on one time-step ahead desires for the CV path [2, 3], as opposed to long horizons of 30 or more future predictions. And usually, the APC products use linear models. The models described here would normally be nonlinear – to solve the problem of a nonlinear process.

The models of the controllers presented here are process-based models, usually employed to accommodate process nonlinearity in processes that have well-behaved dynamic responses. With linear first-order models, PMBC reduces to PI control. In case the model is linear, for simplicity, convenience, and standardization, you should use PI, not PMBC.

In a sense, both fuzzy logic control and expert system control can be considered as nonlinear model based controllers. But, these use linguistic language models that reflect the operator/engineer's opinion of the process behavior. The models in PMBC use first-principles models like those found in textbooks. First-principles models are usually simpler and more precise than linguistic models, and using them within the control system provides secondary benefits.

Many SISO controller products are adaptive or self-tuning. In either case, a supervisory component of the controller observes the process response and adapts the PID tuning coefficients or model coefficients so that the control law tracks the changing process. Usually the internal models are linear, but some products use nonlinear modeling approaches. However, the models are empirical, and do not reflect the engineer's view of the process. In contrast, the approach in this tutorial is to use the process engineer's first-principles models, not to adapt linear models.

The controller structures presented here would not be right for processes in which ill-behaved dynamics (variable deadtime or inverse action) or future constraint consequences are significant dynamic factors. The controllers can be used for processes with any of the following well-behaved dynamics: open loop stable, high-order, integrating, or open loop unstable processes. If ill-behaved dynamics (delay or inverse action) is the primary problem, and nonlinearity is mild, then consider Internal Model Control, IMC [4].

However, similar to APC, the controllers described here can be used within an optimization structure to accommodate extra degrees of freedom situation (DoF>0) (determine the economic best MV values when there are more MVs than CVs), the DoF=0 situation (a square MIMO system) or the DoF<0 situation (where an MV is removed or hits a constraint, and the remaining MVs need to be moved to best balance performance of all CVs).

Although the models described here can also be used within a horizon-predictive, constraint-handling, multivariable controller; only simple implementation of one-step-ahead calculations are developed in this tutorial.

The controllers described here would not normally be classified as ARC (Advanced Regulatory Control) because they are not cascade, ratio, or override structures for PID algorithms or linear dynamic compensators such as feedforward or decouplers. However, SISO GMC [2] and PMBC [3] controllers can become elements within ARC structures.

4. SISO Process-Model Based Control

Simple SISO Process-Model Based Control (no parameter adjustment)

There are three functions within model based control – predict, correct, and act. The structure can be represented by the operations in Figure 1.

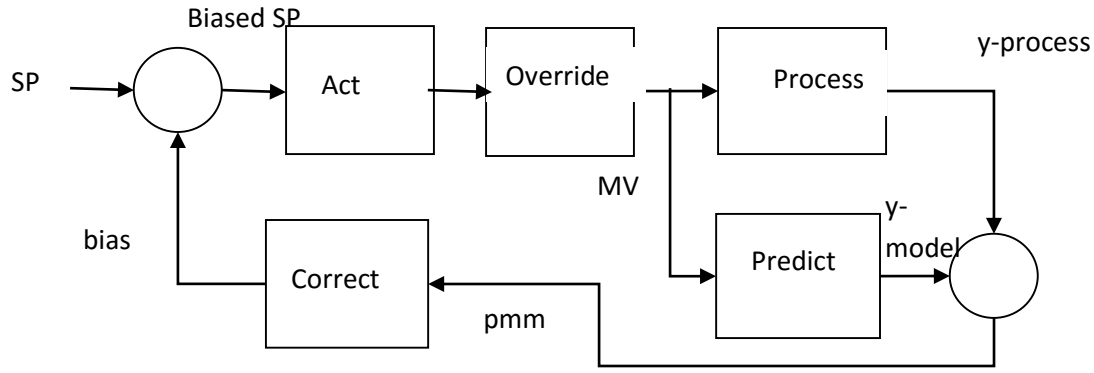


Figure 1 – PMBC Representation

Predict

The first PMBC function is to use the model to predict or mimic what the process output is expected to do. This is the lower right box in Figure 1. This updates the past model value with past disturbance and control action, to predict the updated (current or new) modeled output. With conventional practice of 30 control samplings in an open loop transient (or 10 samplings within a time-constant), the control interval, Δt, is usually small enough to permit Euler's explicit finite difference method to solve the model numerically. Using Equation (18) to represent the SISO dynamic model, this model prediction calculation becomes

$$y_{m,i} = y_{m,i-1} + \Delta t \cdot f(y_m, u, d, p)_{i-1} \tag{23}$$

Where "i" is the time increment counter, representing the present value, and "i-1" the immediate past value. The subscript "m" refers to the modeled value, not the measured process value. However, "u" refers to the actually implemented MV value, after any overrides change what the controller initially desired. The symbol "d" refers to measured disturbances or feedforward variables, and "p" represents parameters values in the model.

However, since computers use assignment statements, the i-subscript is not necessary. The new modeled y-value is based on the old conditions. Here the symbol ":=" will indicate a computer assignment statement.

$$y_m := y_m + \Delta t \cdot f(y_m, u, d, p) \tag{24}$$

Correct

The second function within model-based control is a correction for process-model mismatch. Process model mismatch is the difference between modeled prediction and process measurement, as represented by the circle difference operation on the lower right side of Figure 1. The modeled value will not exactly

match the process value for a variety of reasons including model error (due to either simplification or an inexact parameter value) or unmeasured disturbances. A simple correction approach is to bias the set point with the process model mismatch as follows:

$$pmm = y_p - y_m \tag{25}$$

$$y_{SPbias} = y_{SP} - pmm \tag{26}$$

The logic is, "If you aim at the target, but hit 3 cm low. Then next time aim 3 cm above the target." This is illustrated by subtracting the pmm from the SP as the upper left circle difference operation in Figure 1. The box labeled "correct' in the lower left of Figure 1, could provide other functions (one to be explained later, but in this simple correction approach, it simply is a pass-through of the pmm value.

Act

The third model-based control function is to calculate the control action, the controller MV value. This is represented as the box labeled "act" in Figure 1. This starts with a user-defined, desired performance objective for the controller to achieve. A simple desire is to calculate a u-value that would make the controller push the model toward the biased set point at a rate proportional to the deviation from biased set point. Mathematically this desire is:

$$\left.\frac{dy_m}{dt}\right|_{desired} = \frac{y_{SPbias} - y_m}{\tau_{want}} = Kc(y_{SPbias} - y_m) \tag{27}$$

Where Kc is used as the reciprocal of tau-want so that a larger value of Kc leads to a more aggressive controller action. For some this is more convenient.

Then, matching the desired rate to the modeled rate

$$\left.\frac{dy_m}{dt}\right|_{desired} = f(y,u,d,p) \tag{28}$$

Solve for the value of u:

$$u = f^{-1}\left(\frac{y_{SPbias} - y_m}{\tau_{want}}, y_m, d, p\right) \tag{29}$$

The symbol, $f^{-1}()$, in Equation (29) represents the inverse. Normally, the model of Equation (18) is used to calculate the rate of change of the process response, *dy/dt*, given process influences and state. However, in Equation (29) the solution (mathematical operation) needs to determine the value of the MV, *u*, that would cause a desired rate of change, the inverse of the normal modeling solution procedure.

Each of the three functions (predict, correct, act) could be specified with a variety of options or could be executed with a variety of techniques. The multitude of choices has led to a variety of controller algorithms and associated acronyms. For instance, for *Act*, the desired performance could be for the model to follow a reference trajectory toward the biased set point. For feedback *correct*ion, the pmm

value could correct the model, and then the corrected model could be moved toward the set point. Implicit, semi-implicit, and other methods could be used to solve for the new modeled value. Optimization can be used to solve for u in Equation (28) rather than using the inverse of Equation (29).

There are many acronyms and product names for the many model-based control alternates. Linear ones include: IMC, DMC, HPC, MPC, RMPCT, Connoisseur, IdCom, Brainwave, MRAC, ... Nonlinear ones include: PFC, ADRC, NLIMC, PMBC, FLC, GMC, GE(MVC), Perfector, MFAC, GNN MPC, Nova … A more complete listing can be found in [1].

If the model is linear and first-order, the 3-stage method described in Equations (24) through (29) is equivalent to an internal reset feedback version of a standard PI controller with the integral-time equal to the process time-constant. It is comforting that idealizations lead to a familiar solution, confirming the correctness of the control steps.

However, if the model is linear and first-order (FO), use the conventional PI control technique, which will be comforting to many of your stakeholders.

For the case of the mixing tank (Process Example 2), the calculations in the AUTO mode of a concentration controller would be

Model	$cmodel := cmodel+\Delta t*(F1*c1+F2*c2)/V$	(30)
pmm	$pmm := cmeas-cmodel$	(31)
Bias Sp	$cmodelSP := cSP-pmm$	(32)
Desire	$dcmodeldt := Kc*(cmodelSP-cmodel)$	(33)
Inverse	$F2 := (V* dcmodeldt+F1*(cmodel-c1))/(c2-cmodel)$	(34)

Notably, as an issue for nonlinear control, the set of operations could lead to execution errors. For example, if the value of V in Equation (30) is zero, or if c2=cmodel in Equation (34) a divide-by-zero error prevention logic must override the calculation. However, the override options should be clear to the process engineer. It is not a case of "Well, now what do I do?" If V=0, for example, then there is no mixing lag in Equation (4) and the value of cmodel should be the steady-state value predicted by Equation (3). If V=0 then replace Equation (30) with Equation (3). And, if the modeled tank concentration, cmodel, is equal to the inlet concentration, c2, then either c1 is equal to c2 or F2 had been excessive (or F1 had been zero-ish). In either case, F2 cannot be used to change cmodel; and, as a result F2 should be zero.

Equation (30) should also be used in the MAN mode to update the model so that the model is appropriately initialized for bumpless transfer when switched to the AUTO mode. In the MAN mode, I also like set point tracking, and recommend updating the set point with the measurement.

With these features the Simple SISO PMBC algorithm is

```
IF mode = "MAN"  THEN
        IF V>0 THEN
                cmodel := cmodel+Δt*(F1*c1+F2*c2)/V
        ELSE
                cmodel := (F1*c1+F2*c2)/(F1+F2)
        END IF
        cSP := cmeas
```

```
ELSE                'Controller is in AUTO
        IF V>0 THEN
                cmodel := cmodel+Δt*(F1*c1+F2*c2)/V
        ELSE
                cmodel := (F1*c1+F2*c2)/(F1+F2)
        END IF
        pmm := cmeas-cmodel
        cmodelSP := cSP-pmm
        dcmodeldt := Kc*(cmodelSP-cmodel)
        IF c2>cmodel THEN
                F2 := (V* dcmodeldt+F1*(cmodel-c1))/(c2-cmodel)
        ELSE
                F2 := 0
        END IF
END IF
```

This code will qualify for IEC 61131-3 or 61499 - Structured Text (ST) format. "Structured Text" is a generic representation of vendor-independent high level programming language that can be used for industrial automation. However, this particular code sample, written in Excel VBA, may not meet syntax requirements for your particular implementation.

Necessarily, some variables need to be explicitly initialized. So, in any code there should be a statement such as:

```
IF First_Pass THEN
---
--- Initialize coefficient and variable values here
---
END IF
```

Example 11f – show process nonlinearity, impact of disturbance, controller robustness to functional model mismatch and tuning factor, set point tracking and disturbance rejection, robustness to noise and disturbances. This uses the car model of above Equation (17).

$$\frac{m}{avg_c}\frac{dv}{dt}+v=\frac{k}{av}u-\frac{\sin(\theta)mg}{avg_c}$$

But instead of the simple k*u linear influence of the MV, and a knowable value for angle and air and rolling friction, the model is.

$$\frac{m}{avg_c}\frac{dv}{dt}+v=\frac{ce^{-u/20}+de^{-u/30}}{av}-\frac{b+drag}{avg_c}$$

The simulated process uses similar but different functionality and coefficients.

SISO Process-Model Based Control (with parameter adjustment)

Often there are process features or attributes that change with time (termed non-stationary attributes), and which are indicative of process health, performance, or condition. These include catalyst reactivity coefficient, impurity concentration, heat exchanger fouling factor, friction losses in fluid piping, ambient heat losses, active ingredient in raw material, viable cell growth factor, reaction yield, average tray efficiency, etc. Coefficient values in a model, indicated by "p" in Equation (18), represent those attributes. As the process changes the model parameter values will change. Models with coefficient values that change in time are termed non-stationary models. Process owners monitor the model coefficient values that reflect such factors to inferentially observe process condition, performance, or health; and use that information to trigger events, schedule maintenance, predict operating constraints and bottleneck capacity, and etc.

Once the biased set point for a process model is being updated on-line, in real time, for control; it is a relatively easy addition to also adjust the value of the model parameter that matches that non-stationary process feature. This action not only provides information for the process owner, but also improves control by keeping the model locally and temporally true to the process.

There are many ways to update model parameters, to adapt models in real-time. Again, this has led to a variety of adaptive control methods. A simple and effective approach to adjustment of a first-principles model is to desire that the model parameter be adjusted so that pmm approaches zero in a first-order manner when the process is kept at about a steady state. The mathematical statement of the desire is:

$$\tau_{want-pmm}\frac{dpmm}{dt} + pmm = 0 \tag{35}$$

Choosing to adjust a coefficient that affects the steady state, combining Equations (20) and (25)

$$pmm_i = y_{p,i} - g_i(u,d,p) \tag{36}$$

Substituting into Equation (35) using the chain rule of differentiation

$$\tau_{want-pmm}\frac{\partial[-g(u,d,p)_i]}{\partial p}\frac{dp}{dt}_i + pmm_i = 0 \tag{37}$$

Which, rearranged and using the finite difference approximation for *dp/dt*:

$$p_i = p_{i-1} + \frac{\Delta t \cdot pmm_i}{\tau_{want-pmm}\dfrac{\partial g}{\partial p}_{i-1}} \tag{38}$$

Equation (38) is recognizable as a Newton's method of incremental, recursive parameter updating (root finding), with $\tau_{want-pmm}$ as a tempering factor, which prevents excessive changes in the model parameter value and reduces measurement noise effects. This is very similar to the IMPOL method [5] demonstrated in [6]. In computer assignment statements the subscripts are unnecessary, because the past values are used to assign the new value.

$$p := p + \frac{\Delta t \cdot pmm}{\tau_{want-pmm} \dfrac{\partial g}{\partial p}} \tag{39}$$

The value of $\tau_{want-pmm}$ should be large compared to model dynamics (so that model adjustment does not interact with control calculations), but small compared to the time period over which the process attribute changes (for rapid tracking of the process).

The choice of which model parameter is to be adjusted should be compatible with the following five principles:

1. The model parameter represents a process feature of uncertain value. (If the value could be known or otherwise calculated the method of Equation (39) would not be needed.)
2. The process feature changes in time. (If it did not change in time, adaptation would be unnecessary.)
3. The model parameter value has a significant impact on the modeled u-y relation. (If it has an inconsequential impact, there is no justification to adjust it.)
4. The model coefficient should have a steady state impact, if data are to come from steady state periods. (If it does not, if for instance the coefficient is a time-constant, then when the process is at, or nearly at, a steady state, the model coefficient value will be irrationally adjusted in response to process noise.)
5. Each coefficient to be adjusted needs an independent response variable.

If one or more of the three principles are not true, there is no sense in adapting that model parameter on-line.

As a specific example, again using the mixing tank model, assume that c1 is not measureable, that it changes in time, and is chosen as the model parameter to be incrementally updated. Then Equation (39) becomes.

*c1 :=c1+(cmeas-cmodel)*V/(tauwpmm*F1)* (40)

This incremental model parameter adjustment should be included in both the MAN and AUTO control modes. But, the reader should also recognize that if F1 is zero the calculation cannot be executed. Should F1 be zero, there is no information that can be used to adjust c1 by the process output. Accordingly, there is no justification to attempt to use the method to update the model parameter, c1. The SISO PMBC computer code becomes.

```
IF mode = "MAN"  THEN
        IF V>0 THEN
                cmodel := cmodel+Δt*(F1*c1+F2*c2)/V
        ELSE
                cmodel := (F1*c1+F2*c2)/(F1+F2)
        END IF
        cSP := cmeas
        pmm := cmeas-cmodel
        IF F1 >0 then c1 :=c1+pmm*Δt*(F1+F2)/(tauwpmm*F1)
ELSE            'Controller is in AUTO
```

```
        IF V>0 THEN
                cmodel := cmodel+Δt*(F1*c1+F2*c2)/V
        ELSE
                cmodel := (F1*c1+F2*c2)/(F1+F2)
        END IF
        pmm := cmeas-cmodel
        IF F1 >0 then c1 := c1+pmm*Δt*(F1+F2)/(tauwpmm*F1)
        cmodelSP := cSP-pmm
        dcmodeldt := Kc*(cmodelSP-cmodel)
        IF c2>cmodel THEN
                F2 := (V* dcmodeldt+F1*(cmodel-c1))/(c2-cmodel)
        ELSE
                F2 := 0
        END IF
END IF
```

Example 11f – PMBC with model adjustment. Show removal of pmm, rate of removal influence by tau-pmm, and ability of corrected model to better forecast max value of controller is at 100% (constraint for supervisory optimizer). With noise and ARMA disturbance.

5. Solving the Inverse

The controller must calculate the inverse of the model. For models that are linear in the y-SS response to u, the inverse results in an explicit relation. In general, because there are so many forms of nonlinearity, one cannot claim that there will always be an explicit inverse. For example, the nonlinearity in Process Model Example 4 (Hot and Cold water mixing) is represented by the compound fraction of Equation (11). And, the nonlinearity of Model Example 1 (fluid flow) is the complex of quadratic, reciprocal, and square root of Equation (1).

However, in many cases from my experience, the inverse can be solved explicitly for the MV value. This would be a strong preference within the K.I.S.S. values that direct engineering.

Even if there is an explicit inverse, there may be situations that lead to execution errors, or infeasible solutions. For example, if the desired rate of change for the flow rate response to a deviation from set point is high, then the inverse of Equation (1) could lead to calculation in which the argument of a square root has a negative value. In this case, the override option is obvious, if the desired drop in flow rate is more extreme than any MV action can cause, the best action is set u=0. In my experience, the possible execution error or infeasibility has always been obvious in the equations, and the appropriate override action (either u=0% or u=100%) has also been obvious.

However, I cannot guarantee that every type of nonlinearity will be as kind to a future control engineer and reveal an obvious override action. Desirably, we seek a method to obtain the inverse that is guaranteed to obtain the right result when encountering an infeasible calculation. I will recommend optimization.

But first, consider a classic method of solving an implicit equation by root-finding. Although often practicable, there are additional difficulties in control. Equation (18) represents the dynamic model. If

the rate of change is specified, then the u-value is the unknown. Rather than solving explicitly for u as indicated by Equation (29), rearrange the model as:

$$\frac{dy_m}{dt}_{desired} - f(y_m, u, d, p) = h(\frac{dy_m}{dt}_{desired}, y_m, u, d, p) = 0 \quad (41)$$

Now h() is a function of u, and the desired value of u makes h=0. This is a classic root finding approach, and there are many methods that can be implemented. Common techniques are interval halving (bisection), successive substitution, or one of the Newton's variations (secant, etc.). I prefer interval halving between the u=0% and u=100% extremes. Although successive substitution or a Newton-type approach may converge in fewer iterations, they are subject to function features that could send them in infeasible regions or require an indeterminate number of iterations. If a root exists within the feasible range, then interval halving can be guaranteed to find the root (within the convergence criterion) within a specified number of iterations.

However, interval halving cannot be employed for MIMO cases. Further, if the desired action could lead to a u-value that is beyond the 0% and 100% limits, there may not be a root within the feasible MV range.

As a result, when there is not an explicit inverse, I recommend optimization as the method to determine the u-value.

In the optimization approach, to define the objective function, square h in Equation (50). The optimization statement is:

Min J = [h(dy/dt desired, y, u, d, p)]^2 (42)
{u}
S.T. 0≤u≤100

This reads, "Determine the value of u that minimizes the objective function, J (=h-squared), subject to the limits on the u-value. In optimization the variable being changed (in this case the MV, u) is termed the decision variable (DV). Equation (42) is the statement, not the solution. To solve an optimization problem, you need to use one of many methods. For a single DV application the optimization method could be Golden Section or Newton's. For MIMO situations optimization methods are Nelder-Mead, Hook-Jeeves, Levenberg-Marquardt, and many others.

The notation for the optimal u-value is u*. If there is a feasible root for Equation (41), then there is a u-value that makes the h-value, and consequentially the h-squared value, equal to zero. Any deviation from this u*-value leads to a negative or positive h-value, which when squared is a positive value, making the Equation (42) u* = root u-value for Equation (41).

Although gradient-based optimization methods use fewer iterations to find u* than do direct search methods when the J response to u is quadratic-ish, direct search methods often use fewer function evaluations and are not confounded by surface aberrations that might arise from the nonlinear equations. I recommend direct search approaches. For a SISO application Golden Section would be a nice choice; however, the optimizer needs to be able to accommodate infeasible trial solutions which might arise if a desired rate of change or disturbance value leads to an execution error. Further, it is possible that a combination of situations could lead to multiple optima on the J(u) curve (although I have only personally

experienced this in situations that were contrived to make it happen). Accordingly, I prefer multi-individual, or multi-particle, or multi-player optimizers such as particle swarm, differential evolution, or Leapfrogging. These direct search approaches can handle infeasible regions, multiple optima, and are not confounded by surface aberrations and constraints. Of these, I prefer Leapfrogging for its combination of simplicity, understandability, and technical performance [7].

However, as an introduction here, for simplicity and clarity of method, first, I am using a heuristic cyclic search as the optimizer.

6. Cyclic Heuristic Direct Search

In the single decision variable (DV), one-dimensional version of this optimization search, start with a single feasible trial solution, a DV value, and evaluate the OF value. This is the base point. Then increment the DV by ΔDV, and evaluate the OF. If the new trial point is both feasible and has a better OF value than the base point, then you are moving in the right direction. So, 1) make the new trial point the base point, and 2) increase the ΔDV value so that the next time, you make a bigger step. Alternately, if the new trial point is either infeasible or does not have a better OF, then you likely moved in a wrong direction. So, 1) retain the prior base point, and 2) reverse sign and contract the ΔDV value. Repeat until convergence criteria is met.

In the N-Dimensional version, there are multiple decision variables, multiple controller MVs. In this optimization search, start with a single feasible trial solution and evaluate the OF value. This is the base point. Then increment the first DV by ΔDV_1, keeping the other DVs at their base values, and evaluate the OF. If the new trial point is both feasible and has a better OF value than the base point, then you are moving in the right direction. So, 1) make the new trial point the base point, and 2) increase the ΔDV_1 value so that the next time you explore that direction, you make a bigger step. Alternately, if the new trial point is either infeasible or does not have a better OF, then you likely moved in a wrong direction. So, 1) retain the prior base point, and 2) reverse sign and contract the ΔDV_1 value. Repeat this for each DV, one-by-one. When a change in each DV has been explored, test for the convergence criteria. Each cycle through all DVs is an iteration.

I recommend values of 1.05 to 1.2 for the expansion factor. If moving in the right direction, each iteration moves 5% to 20% further than the past. Larger expansion values make the exploration point too aggressively moved for difficult surfaces after a few successes. Smaller values increase the number of iterations to get to the optimum.

I recommend values of 0.5 to 0.8 for the contraction factor. These heuristic values also arise from qualitative arguments.

The user must choose the initial ΔDV_i values. I recommend starting with 10% of the DV range. If the wrong size or sign, contraction or expansion will quickly adjust the ΔDV_i values.

The algorithm is very robust, effective, and relatively efficient, and code is very simple. Here is the active section of code:

```
For i = 1 to Number_of_DVs
    u(i) = ub(i) + du(i)                    'Increment i^th DV to search what happens.
```

```
        CALL ConstraintTest                 'Determine if hard constraints are satisfied.
        IF Constraint = "PASS" THEN         'No hard constraints are violated.  Yea!
                Call ObjectiveFunction      'Determine the OF value.
                IF OF < OFb AND Constraint = "PASS" THEN    'If good, then ...
                        ub(i) = u(i)        '... accept the new value for u(i), ...
                        OFb = OF                    '... update the new best SSD value, and ...
                        du(i) = 1.2 * du(i) '... make the next change larger.
                ELSE                                'Otherwise (equal or worse OF), then …
                       u(i) = ub(i)         '... reset the u(i) to the base case, and ...
                        du(i) = -0.5 * du(i) '... reverse and contract search increment.
                END IF
        ELSE                                'Constraint was violated, trial solution was bad.
                u(i) = ub(i)            'Reset the u(i) to the base case, and ...
        du(i) = -0.5 * du(i)            '... reverse and contract the search increment
        END IF
NEXT i
```

Optimizers need an initial value(s) for the initial trial solution(s). In my experience using the past solution (the prior optimal MV value) as the initial guess is best. It is the correct new solution if there is no change, and very close to the correct new solution during a transient. Starting close to the prior optimal solution means fewer iterations to converge.

7. Leapfrogging Search

This is a multi-player direct search. Contrasting the traditional optimizers that start at a single trial solution and move it in the downhill direction, multi-player approaches start by randomly populating the MV space with players (use multiple initial trial solutions). Some are higher on the OF curve than others. The worst (highest) leaps over the best into a random spot in the reflected window, in the DV space on the other side of the best. This is a direct search method that has the advantage of a higher probability of finding the global minimum. When the best is far from the minimum, the cluster leapfrogs toward the minimum. When the best is in the vicinity of the minimum, the cluster converges.

```
Call Initialize_Players    '  Initialize player positions and OF value
Call Find_Best             '  Search for highest and lowest player
Call Find_Worst
For Iteration = 1 To 500            'up to 500 iterations - it seems that 50 is plenty
    dx = xlow - xhigh           'difference between players with highest and lowest OF values
    constraint = "Unassessed"
    Do Until constraint = "PASS"    'each must be in a feasible location, repeat if not
       xhigh = xlow + Rnd() * dx   'high (or infeasible) jumps to random position in window
       pf(Nhigh) = f_of_u(xhigh)   'calc new OF for relocated trial solution
       If constraint = "FAIL" Then          'each player must be in a feasible location.  If not, then
          dx = xlow - xhigh      'reset difference between trial solutions
       End If
    Loop
    px(Nhigh) = xhigh       'reassign position of former high player to its new feasible location
   If pf(Nhigh) < pf(Nlow) Then      'If new player is better than former best, reassign the low
```

```
      Nlow = Nhigh
      xlow = xhigh
    End If
    Call Find_Worst              '  Re-search for worst player
    Call Find_xRange
    If xRange < 0.1 Then Exit For        'Stop iterations if range between players is small
Next Iteration
```

LF is a bit more involved than the cyclic direct, but still relatively simple with respect to other multi-player direct search optimizers, and LF is very effective and relatively fast in higher dimensions for constrained, multi-optima, discontinuous, nonanalytic functions. For well behaved (unconstrained, quadratic-like, continuous differentiable) functions, gradient based optimizers work best. However, for the aberrations that result in nonlinear control applications, I like LF.

8. Non-First-Order Dynamics

The PMBC approach can be extended to use process models that are high-order, open-loop unstable, and integrating. The extrapolation of the technique presented in Equations (30)-(34) is straightforward; however, the explicit presentation is beyond the time and page constraints of this work.

If either process deadtime or inverse action is a major difficulty, then PMBC can be based on one of several simple modifications to the *Act* concept. Either 1) desire that the model match the biased set point at a coincidence point in the future (which is kin to Predictive Functional Control), or 2) desire that it find a u-value to make the process best follow a reference trajectory (which is kin to the CV damping version of model predictive control), or 3) limit the inverse to the model without the delay or inverse dynamics (after the fashion of Internal Model Control). Again, these are straightforward, but beyond the scope of this tutorial. My preference is for option 2) because it is kin to MPC and easily adapted to find a sequence of future MV values to accommodate for future constraints.

9. MIMO Model-Based Control

In addition to nonlinearity, MIMO systems express interaction, constraint situations of degrees-of-freedom DoF<0, and extra MV situations of DoF>0.

The first aspects, nonlinearity and interaction, are each revealed in Equations (11) and (12) of Model Example 4, mixing of hot and cold water. Nonlinearity means that a process output is not a linear response to a process input. For example, in Equation (11) T is a nonlinear response to both u1 and u2. Interaction means that process inputs affect more than one process output. If either u1 or u2 change, both F_{total} and T_{mix} respond.

A process-model based controller would specify desired individual rates of change for the modeled T and F; and, with the modeled values of T and F updated, this would leave Equations (11) and (12) as two equations and two unknowns. For control, solve the set of equations for the two unknowns, u1 and u2. In this specific case, it is possible to solve for u1 and u2 explicitly. However, in certain cases, it may not

be possible to solve for the inverse explicitly, in which case a root-finding approach could be used. However, instead of root-finding, this tutorial recommends optimization because it can handle hard constraints on excessive MV values, the cases of DoF>0 and DoF<0, and a penalty when auxiliary variables exceed limits.

For MIMO systems the optimization statement is:

$$\text{Min} \quad J = \sum_i (h_i / EC_i)^2 + \sum_j B_j / EC_j + \sum_k c_k u_k / EC_k \tag{43}$$

$\{u_k\}$
S.T. $0 \le u_k \le 100$

In this statement, *h* is the contribution to the objective function that represents the controlled process performance desire, such as that in Equation (41). The symbol *B* represents a penalty for "badness", an excess or constraint violation. And, the coefficient *c* represents the cost associated with each MV. The equal concern factors, *EC*, are user-defined factors that balance the importance of each item. Some people prefer to show the normalizing coefficients as multipliers (Lagrange coefficients, or weighting factors).

The process parameter adjustment approach can also be extended to multivariable processes. If there are the same number of measured process outputs as adjustable parameters, then Equation (38) becomes a matrix-vector equation:

$$\left[\frac{\partial f_i}{\partial c_j}\right]\left[\Delta c_j\right] = \left[\, pmm_i / \tau_{want-pmm,i} \,\right] \tag{44}$$

The matrix of partial derivatives (the Jacobean, or sensitivity matrix) must be invertible – it must have a non-zero determinant. For this, several conditions must be true: First, no row can have all zero elements. Such an event would mean that none of the adjustable parameters have an impact on a particular modeled output. Second, no column can have all zero elements. Such an event would mean that one of the adjustable parameters has no impact on any of the modeled outputs. Checking for these should be intuitively easy. Finally, third, none of the rows or columns can be linearly dependent on other rows or columns. Such an event could mean that interaction between parameters chosen to be adjustable cannot be separated.

However, if there are fewer number of adjustable parameters than process measurements, then Equation (44) can be reformulated as a least squares minimization. Additionally, there are many options to model adaptation and filtering which the user might prefer.

10. Implementations

These controllers have been demonstrated with success on commercial-scale, pilot-scale, and lab-scale processes. SISO applications include control of fluid flow rate, heat exchanger temperature, distillation bottoms composition, plasma reactor pressure, and pH. MIMO applications include distillation dual end composition control, fluidized bed gasification [8-27].

Paper 040-00221 in the 2011 ISA Automation Week Conference [8] demonstrates this SISO PMBC method on differential pressure control of a packed tower absorber. It evolved to a cascaded PMBC-to-PMBC controller and is published in ISA Transactions 2013 [28]. The column dP is modeled as a Darcy's law response to gas flow rate with density being dependent on the average column pressure (which is dP dependent). The packed column resistance to flow, drag coefficient, depends on the liquid absorbent flow through the column, which changes in time, creates a process feature that is too complex to model effectively, and which has a significant impact on dP. The authors show that the process is nonlinear, control is effective, and that model parameter adjustment provides legitimate and reproducible insight as to the process behavior.

Raul, et al. [29], demonstrate it on a pilot scale heat exchanger unit.

11. Sample Demonstrations

I have chosen the hot and cold water mixing process as a simulation demonstration for PMBC. The process is more complicated than the model represented in Equations (13) and (14), and nonlinear in many aspects. Details are in Appendix A. Here is a summary of features: First, the flow rates in the simulator are not linear functions of the controller outputs – the valves have a parabolic inherent characteristic (modified equal %) installed in a process line. Accordingly the flow rate response is like that described in Equation (1). Further, since the system pressure drop experienced by either fluid is dependent on the flow rate of the other fluid through the common discharge pipe, the ΔP in Equation (1) depends on the other fluid flow rate. This also makes the total flow rate a nonlinear and interactive response. Additionally, as the flow rate changes, the residence time in the discharge pipe from the mix point to the temperature sensor changes, causing the transport delay (dead time) to change. Further, the dynamics of the heat transfer from the flowing fluid to the thermowell depend on the heat transfer coefficient, which is modeled as a 0.8^{th} power law function of the fluid flow rate. Additionally, the process is fifth-order plus delay, with dynamics associated with valve response, fluid acceleration, transport delay and sensor response. Finally, the process includes environmental disturbances of several forms. There are drifts in the input values for the two fluid temperatures and the two inlet pressures. And, both the temperature and flow rate measurement devices include calibration drifts and measurement noise.

By contrast, the simple first-principles (elementary) model that will be used for control is the first-order model of Equations (13) and (14) – linear and not interactive in flow rate response to controller outputs, and simplistically nonlinear in the temperature response to controller outputs.

Although the model of Equations (13) and (14) can be explicitly rearranged to solve for u_1 and u_2, given a desired rates of change and current modeled values for T and F, the simulator uses optimization. This provides the exact values as an explicit solution when the solution is feasible, but is capable of using the unconstrained MV to balance deviations from the two CVs when a constraint is encountered.

Illustrative simulation experiments will reveal the impact of tuning, model mismatch, set point tracking, disturbance rejection, robustness to environmental issues, constraint handling, model adaptation, and the impact of Equal Concern factors that set the T w.r.t. F priority. The simulations will demonstrate robustness to model coefficient choices, and environmental effects, ease of tuning, zero steady-state offset, constraint handling, and no wind-up on constraints.

25a, Sample Demonstrations on the hot and cold in-pipeline mixing.

12. Insight

The author provides the following observations to help those implementing PMBC:

1. In tuning PMBC, tau-want should be a bit less (80% perhaps) than the dominant process-model time-constant. This makes the controller push the process slightly faster than the process would normally respond.
2. When adjusting model parameter values, tau-pmm, the time-constant for the model parameter adjustment should be about 10 times the dominant model time-constant. This tempers noise and reduces any interaction that might arise between controller and model adaptation.
3. Model parameter adjustment can provide insight on the process behavior for maintenance, health, condition, future constraints, etc. This secondary benefit from a control viewpoint might actually be the primary reason from a maintenance or operations viewpoint.
4. Model parameter adjustment is not necessary for control. The feedback mechanism for control is to bias the set point with the process-model mismatch.
5. Supervisory optimizers use steady state process models to determine economic best operating regions. The optimizer needs to understand constraint possibilities and the temporal state of the process. If the controller models are adjusted in real-time to reflect the temporal process conditions, then integrating the controller models (and model parameter values) within the plant-wide optimizer will provide better set point values and constraint recognition.
6. Use of process models in the controllers preserves and propagates process knowledge among the operational staff. This enhances process understanding, rational trouble shooting, and process improvement; and provides another secondary benefit.
7. The case-by-case model development and controller design is a disadvantage. Each case introduces unique challenges in structuring and implementing the equations to present execution errors and to guarantee a solution. However, often this is easy, but it takes engineering attention.
8. Although my experience finds that local or temporal linearizing these controllers always reveals that they are stable, nonlinearity means an inability to globally guarantee convergence, stability, etc. Some will object to the absence of a universal proof of stability.
9. The inverse, Equation (17), might not provide an explicit solution. Use optimization (as opposed to root finding) when either the controller has multiple MVs or the solution for MV value cannot be from an explicit rearrangement. Optimization is the best way to handle the DoF>0 and DoF<0 situations.
10. You can use model parameter tracking independent of process control.
11. Nonlinear control ability relieves a constraint on process design, permitting design to focus on more competitive outcomes.
12. Implement the controller in stages. Be sure that each stage works before letting it reveal its control decisions. Start with exploring the model off-line to test it for compliance to the process. Then simulate the controller off-line to ensure all execution errors are compensated in MAN to AUTO switches, startup, etc. Then model operating in MAN mode, shadowing the process. If the model needs fixing do it. Fix the model before it is used to suggest control action. Don't let a bad model lead to control action that creates a bad reputation for the project. After there is confidence in the model, have the controller operate in AUTO mode and suggest control action. You will need to use the actually implemented MV values, not the controller-suggested output, as controller model inputs. After there is confidence in the controller, place it on line.

13. The model for the controller must be developed and validated. This takes more time than retuning a controller. But, it may save many retuning instances, and much cumulative re-tuning time in the long run.
14. If the process changes significantly (reconfiguration, product type) then the model may have to be changed.
15. Implementation and diagnosis of such controllers requires a process engineer with several years of control experience.
16. If the control problem is related to process faults (dropped signals, sticktion, analyzer delay, undersized pumps, ill-sized orifices, etc.) then nonlinear control is not the solution.
17. Since the actual MV is used to calculate the modeled value (the overridden MV value in safety or select control, not the prior calculated MV), there is no windup on a constraint.

13. Candidate Processes

1. Nonlinear or non-stationary – A candidate process could be recognized if tuning the controller is a continual problem because the process gain and/or time-constant and/or interactions change with operating region and/or they change in time due to process reconfigurations, product type, flow rate, or input material properties.
2. Condition monitoring – Use the simple models with model parameter adjustment. This provides information related to process condition and constraints.
3. Simple dynamics – This one-step-ahead control action is not appropriate for processes with ill-behaved dynamics (inverse action or significant deadtime) or for processes in which action now could lead to a future constraint violation. However, the approach can be extended to horizon-predictive, constraint avoidance control.
4. MISO – this approach was developed for a single CV and MV with, possibly, measured disturbances. However, it can be applied within a MIMO situation.

14. Cautions

1. Include MAN and AUTO modes with bumpless transfer features – set point tracking and model updating in MAN.
2. Initialize all coefficients and variables in the First_Pass call of the subroutine.
3. Identify and isolate execution errors. Seek opportunities for divide by zero, overflow (real or integer), square root or log of a negative number, type mismatch “O” entered for “0”, missing input, etc. Rearrange equations to prevent such errors, or bypass the calculation if such an event happens. The logical option should be clear. Be sure that variables are appropriately initialized in a power-up. There are many forms for nonlinear equations, which creates many, and not obvious possibilities for execution errors.
4. Check for rational values. For instance, when a model parameter is adjusted (efficiency, reactivity, fouling, resistance, inlet composition, reactivity, yield) values should be non-negative.
5. If there is an external override, use the overridden MV value within the controller model (kin to external reset feedback).
6. K.I.S.S. Prior to implementation “Keep it simple, and safe”, so that afterwards you don’t chastise yourself with “Keep it simple, Stupid!” Use a more complicated model only after your determine that the simpler one will not work. The objective is sufficiency, not perfection.

7. Data needs to be valid. The greater the number of process measurements used within the model, the greater the likelihood that there will be a sensor or data transmission fault. Some data faults can cause execution errors. Minimize the reliance on possibly faulty data. Ensure data integrity. Include fault detection or error isolation in the code.
8. To permit Euler's forward finite difference approximation to be valid, the control interval (and simulation interval) should be less than about $1/10^{th}$ of the smallest time-constant in the process or controller.
9. Test the controller by model shadowing in the MAN mode. Only go to AUTO when fully confident that the model, as implemented, is acceptable. Murphy's Law applies to control. If it doesn't work at 3:00 AM you'll get a call, and may not be able to implement another model-based controller.
10. Preferentially, solve for the inverse by rearrangement of the model equations to provide explicit values for the MV.
11. The model inverse and model must have identical gains. It is possible to obtain separate empirical SS relations for model and inverse with one data set. But, the regression slope of y vs. x is not usually the exact reciprocal of the regression slope of x vs. y.
12. If the inverse is solved by an optimizer, it needs to handle a nonlinear constrained case with the possibility of local optima traps. Use a direct search approach to avoid the misdirection that can come from gradient based methods, and to permit hard constraints. Thoroughly explore the optimization surface, and where possible, structure the equations so that there is only ever one minimum. If there is the possibly of local minima, use a multi-player (multi-particle, multi-individual) optimizer to increase the probability of finding the global minimum. Apply convergence criteria to the decision variable and set it about $1/10^{th}$ of the smallest detectable MV change.
13. The model for the controller must be developed and validated. This takes more time than retuning a controller. But, it may save many retuning instances, and much cumulative re-tuning time in the long run.
14. If the process changes significantly (reconfiguration, product type) then the model may have to be changed.
15. Implementation and diagnosis of such controllers requires a process engineer with several years of control experience.
16. MIMO, SIMO, or MISO applications may be a big step in complexity over SISO applications, if they use either 1) optimization to handle constrained (DoF<0) or excess MV (DoF>0) situations, or 2) data validation/correction techniques to handle data faults.
17. If the control problem is related to process faults (dropped signals, sticktion, analyzer delay, undersized pumps, ill-sized orifices, etc.) then nonlinear control is not the solution.

15. Q&A

Q: How does his PMBC technique differ from other model based control techniques that readers may already be familiar with (e.g., Model Reference Adaptive Control, Smith Predictors, etc.)?

A: It seems in the 70s and 80s many people realized the potential for computers to do more than PID, and independently conceived "hundreds" of model-based control approaches. Basic concept elements were independently conceived, but with many nuances for each stage in the control algorithm function. In my experience, with the right limiting assumptions, one controller becomes another. GMC with a Steady

State nonlinear model is the same as output characterization from a PI controller. With a linear model is it simply PI. Internal Model Control (IMC) with a second-order model and a desire that it track a first-order response to a set point change leads to a PID Controller. If the process were linear, first-order, with a fixed delay, then PMBC would reduce to a Smith Predictor, as does IMC. The functions within Advanced Process Control (APC) or the academic community term for it, Model Predictive Control (MPC), can be expressed as Figure 1; and, in a limiting case of MISO single step ahead, reduce to IMC. Classically, the Smith Predictor uses a FOPDT model of the process to account for the deadtime, and the process-model mismatch to bias the set point for a PI controller. PMBC, IMC, and other model-based controllers would have a similar structure, but the "Act' function in Figure 1 would be the model inverse not PI. I have not done much with MRAC, I've been more interested in the use of nonlinear models in control and all my touches on MRAC showed people using linear models. But, I suspect with the right choices, one can be converted to another.

Q: The examples in the article focus on non-linear processes, where the non-linearity is in the process itself. Would PMBC also be used for situations where the process itself is linear, and only the control valve is nonlinear - or are simpler methods OK for this situation?

A: I do promote the use of the best tool for the situation. Best could mean simplest, most convenient, standard, conventional, least expensive, etc. PMBC preserves process knowledge and has only one tuning coefficient. But, if PMBC is low on the learning curve, and gain scheduling or output characterization are expedient, or understood, then they would be the best choices. A nonlinear valve could be handled by either gain scheduling or output characterization. So, could many nonlinear processes.

Q: Suppose deadtime is the problem. Will PMBC work?

A: Yes, PMBC will work. But if the process is linear, either IMC or a Smith Predictor may work as well, and might have the familiarity comfort that would make them preferred. As with those methods, if the process transport delays vary, and the controller deadtime is fixed, then control can go "bad". With a process-based model, the PMBC approach would have the delay based on the process conditions. The ill-behaved dynamics of delay, inverse action, and very high-order were not addressed in the article. If a process has deadtime, then it is not feasible to expect to be able to move it instantaneously. The appropriate controller desire would be to start the process moving at the desired rate after the delay.

Q: Can PMBC be used in a cascade, ratio, or override situations?

A: Yes. The PMBC approach shown here is meant to replace SISO PID controllers that need to be continually retuned because of process nonlinearity. PMBC could be used as primary or secondary controllers, either as both or mixed with PID. I recommend keeping a stage-by-stage design of simple controllers, each with the same Figure 1 form, rather than trying to generate a single code that does it all-in-one. The signal to the model in PMBC needs to be the actual, selected, or overridden value; kin to external reset feedback used to prevent windup when PID is in an override or select situation.

16. Acknowledgments

I appreciate the review and feedback from Bill Poe (Invensys), and Nagappan Muthiah (Contech Control Services). I appreciate the energy and support of my coauthors on the referenced papers in exploring the control possibilities. I appreciate the Edward E. and Helen Turner Bartlett Foundation and the BP/Amoco Foundation for partial support of the activities.

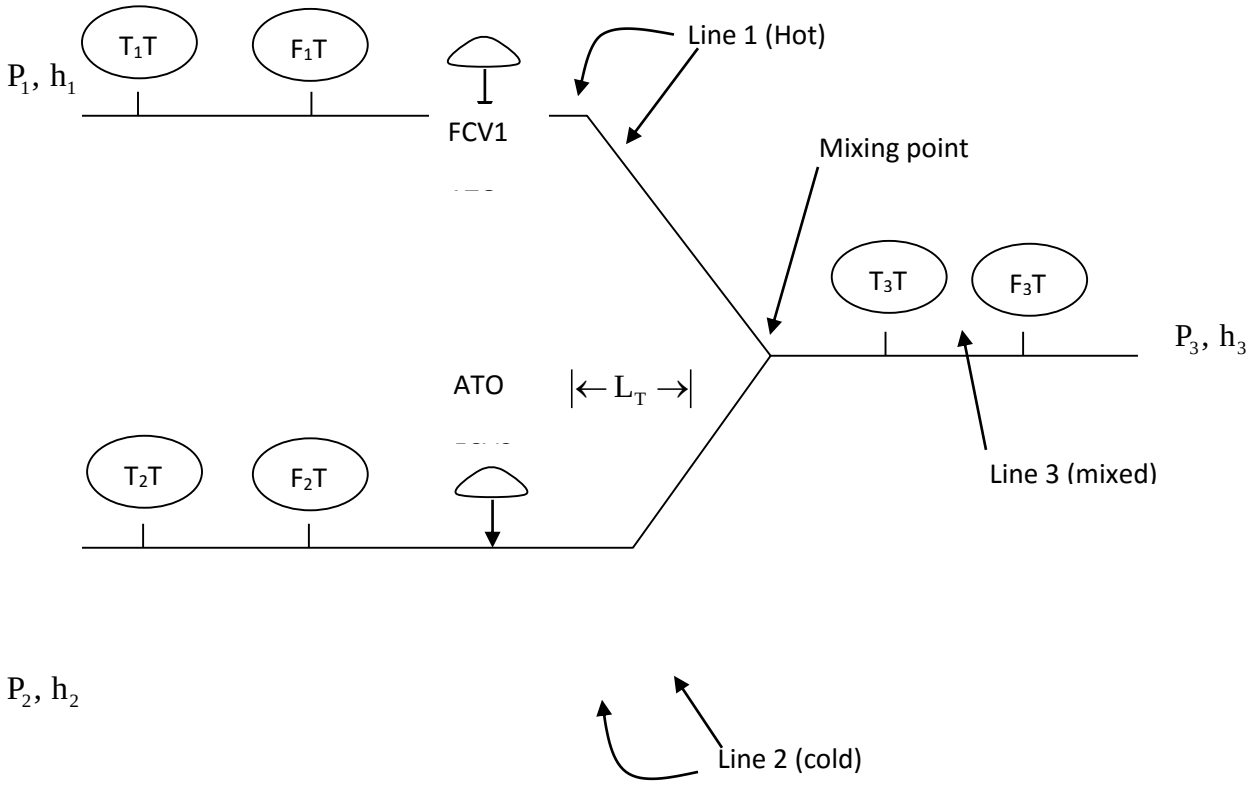
17. References and Resources

1. Rhinehart, R. R., M. L. Darby, and H. L. Wade, “Editorial – Choosing Advanced Control”, ISA Transactions: The Journal of Automation, Vol. 50, No. 1, 2011, pp 2-10.
2. Lee, P. L., and G. R. Sullivan, “Generic Model Control”, Computers and Chemical Engineering, 1998, Vol. 12, No. 6, PP 573-580. [Initial GMC paper]
3. Rhinehart, R.R., and J.B. Riggs, "Process Control Through Nonlinear Modeling,” Control, Vol. III, No. 7, July 1990, pp. 86-90. [PMBC Concept]
4. Garcia, C. E., and M. Morari, “Internal Model Control I. A Unifying Review and Some New Results”, Industrial and Engineering Chemistry Process Design and Development, 1986, Vol. 21, pp. 308-323. [Landmark IMC paper]
5. Rhinehart, R. R., and J .B. Riggs, "Two Simple Methods for On-Line Incremental Model Parameterization," Computers & Chemical Engineering, Vol. 15, No. 3, 1991, pp. 181-189. [On-line model adaptation]
6. Mahuli, S. K., R. R. Rhinehart, and J. B. Riggs, "pH Control Using a Statistical Technique for Continuous On-Line Model Adaptation," Computers & Chemical Engineering, Vol. 17, No 4, 1993, pp 309-317. [PMBC pH control]
7. Rhinehart, R. R., M. Su, and U. M. Sridhar, “Leapfrogging and Synoptic Leapfrogging: a new optimization approach”, Submitted to Computers & Chemical Engineering, January 5, 2011, in revision August 2011.
8. Skach, A., P. Raul, and R. R. Rhinehart, “Process-Model Based Differential Pressure Control on a Packed Tower Absorber”, Submission 0040-000221, Proceedings of the ISA Automation Week Conference, Mobile, AL, October 17-20, 2011. [PMBC control of column differential pressure]
9. Joshi, N. V., P. Murugan, and R. R. Rhinehart, “Experimental Comparison of Control Strategies,” Control Engineering Practice, Vol. 5, No. 7, 1997, pp. 885-896. [experimental comparisons]
10. Subawalla, H., V. P. Paruchurri, A. Gupta, H. G. Pandit and R. R. Rhinehart, “Comparison of Model-Based and Conventional Control: A Summary of Experimental Results,” Industrial and Engineering Chemistry Research, Vol. 35, No. 10, October, 1996, pp. 3547-3559. [experimental comparisons]
11. Mahuli, S. K., and R. R. Rhinehart, and J. B. Riggs, "Nonlinear Model-Based In-Line Control of Wastewater pH: A Laboratory Study," ISA Transactions, Vol. 32, 1993, pp. 241-245. [PMBC pH control]
12. Mahuli, S. K., R. R. Rhinehart, and J. B. Riggs, "Experimental Demonstration of Nonlinear Model-Based In-Line Control of pH," Journal of Process Control, Vol. 2, No.3, 1992, pp 145-153. [PMBC pH control]
13. Dutta, P., and R.R., Rhinehart, "Experimental Comparison of a Novel, Simple, Neural Network Controller and Linear Model Based Controllers," Proceedings of the 1995 American Control Conference, June 21-23, 1995, Seattle, WA, paper TA10-3. [PMBC distillation control]
14. Gupta, A., and R.R. Rhinehart, "Experimental Comparison of Advanced Control Techniques on a Lab-Scale Distillation Column," Proceedings of the 1995 American Control Conference, June 21-23, 1995, Seattle, WA, paper FA10-4. [PMBC Distillation Control]
15. Paruchurri, V.P., and R.R. Rhinehart, "Model-Based Flow Control Boosts Accuracy, Eases Tuning," InTECH Vol. 42, No 4, April, 1995, pp. 52-56. [PMBC flow rate control]

16. Subawalla, H., and R. R. Rhinehart, "Experimental Comparison of Model-Based and Conventional Pressure Control for a Plasma Reactor," 1994 ACC Conference Proceedings, Baltimore, MD, June, 1994, paper FM14 - 1:30, pp. 3122-3126. [PMBC reactor pressure control]
17. Paruchurri, V.P., and R.R. Rhinehart, "Experimental Demonstration of Nonlinear Model-Based Control of a Heat Exchanger," 1994 ACC Conference Proceedings, Baltimore, MD, June, 1994, paper FP14-5:20, pp. 3533-3537. [PMBC Temperature control]
18. Paruchuri, V. P., and R.R. Rhinehart "Experimental Demonstration of a Process-Model-Based Flow Controller," Advances in Instrumentation and Control, Vol. 48, Part 2, 1993, pp. 529-540, Proceedings of the ISA/93 Technical Conference, Chicago, IL, September, 1993. {PMBC flow rate control]
19. Mahuli, S.K., R.R. Rhinehart, and J.B. Riggs, "Nonlinear Model-Based In-Line Control of pH: An Experimental Study," Advances in Instrumentation and Control, Instrument Society of America, Vol. 47, October 1992. [PMBC pH control]
20. Pandit, H.G., R.R. Rhinehart, and J.B. Riggs, "Experimental Demonstration of Nonlinear Model-Based Control of a Nonideal Binary Distillation Column" Proceedings of the American Control Conference, ACC92, Chicago, IL, June, 1992, pp. 625-629. [PMBC Distillation Control]
21. Pandit, H.G., and R.R. Rhinehart, "Experimental Demonstration of Constrained Process-Model-Based Control of a Nonideal Distillation Column," Proceedings of the American Control Conference, ACC92, Chicago, IL, June, 1992, pp.630-631. [PMBC Distillation Control]
22. Rhinehart, R.R., R.M. Felder and J.K. Ferrell, "Process-Model Based Control of a Pilot-Scale Fluidized Bed Coal Gasification Reactor," Advances in Instrumentation and Control, Vol. 43, Part 1, October 1988, pp. 341-349. {PMBC reactor control]
23. Riggs, J. B., and R. R. Rhinehart, "Method for pH Control," US Patent, 4,940,551, July 10, 1990. [PMBC pH control]
24. Guo, B., A. Jiang, X. Hua, and A. Jutan, “Nonlinear adaptive control for multivariable chemical processes,” Chemical Engineering Science 56 (2001) 6781–6791. [GMC Reaction Control]
25. Montandon, A. G., R. M. Borges, and H. M. Henrique, “Experimental application of a neural constrained model predictive controller based on reference system”, Lat. Am. appl. res. vol.38 no.1 Bahía Blanca Jan. 2008. [GMC pH Control]
26. Lee, P. L., R. B. Newell, G. R. Sullivan, “Generic model control — A case study”, The Canadian Journal of Chemical Engineering, Volume 67, Issue 3, pp. 478–484, June 1989 [GMC of an evaporator]
27. Ramchandran, S. B., “Consider Steady State Models for Process Control,” Chemical Engineering Progress, Vol. 93, No. 2, February, 1998, pp. 75-81. [GMC commercial scale distillation]
28. Govindarajan, A., S. K. Jayaraman, V. Sethuraman, P. R. Raul, and R. R. Rhinehart, “Cascaded Process Model Based Control: Packed Absorption Column Application”, ISA Transactions, Vol. 53, No. 2, 2014, 391-401.
29. Raul, P. R., H. Srinivasan, S. Kulkarni, M. Shokrian, G. Shrivastava, and R. R. Rhinehart, “Comparison of Model-Based and Conventional Controllers on a Pilot-Scale Heat Exchanger” ISA Transactions, Vol. 52, No. 3, 2013, pp. 391-405

Appendix A – Hot and Cold Water Mixing Simulator

The simulated process is hot and cold water mixing in a pipeline.



Hot water enters through Line 1 from a header at pressure, P_1, and height, h_1. Cold water through Line 2, from a header at P_2 and h_2. Measured temperatures and flow rates are transmitted to the control room as T1 and T2, and F1 and F2. Flow control valves FCV1 and FCV2 are both air-to-open. The hot and cold fluid meet at the mixing point where they enter Line 3 and exit at P_3 and h_3. Mixed fluid temperature is measure by the sensor-transmitter (T3T) at a distance L_T downstream from the mixing point. Mixed flow rate and temperature are transmitted as F3 andT3.

All transmitters lie - The true process value is biased by calibration drift and noise. Those are normal conditions. Sensor faults are not included in the simulator. Inlet and exit pressures, P1, P2, and P3 vary in time, which makes the flow rates vary even with no change in the signals to the control valves. And, inlet temperatures also vary in time. These are all normal environmental influences.

System dimensions and nominal values for variables are:

Radius = 0.02 m	P_1 = 180 kP_a	$f_1(s_1) = s_1^2$
h_1 = 4 m	P_2 = 210 kP_a	$f_2(s_2) = s_2^2$
h_2 = 1 m	P_3 = 150 kP_a	$\dot{m}_1 = 20\ kg/min$
h_3 = 2 m	L_T = 1.06 m	$\dot{m}_2 = 10\ kg/min$
$Lequ_1$ = 200 m	L_1 = 50 m	$T_{1\ in}$ = 80°C
$Lequ_2$ = 300 m	L_2 = 20 m	$T_{2\ in}$ = 10°C
$Lequ_3$ = 600 m	L_3 = 200 m	ρ = 1000 kg/m^3
Cv_1 = 6 x 10^{-6} m^3/s @ 1 P_a	$\tau_{V1} = 1$ s	μ = 0.001 Pa · s
Cv_2 = 3 x 10^{-6} m^3/s @ 1 P_a	$\tau_{V2} = 1.5$ s	g = 9.80665 m/s^2
$\tau_{T1} = 6$s	$\tau_{T2} = 4$ s	$\tau_{T3} = 3$ s
∈/D = 0.0004	g_c = 1 kg · m / N · s^2	

L_{equi} are the equivalent length of Lines 1, 2, and 3 due to devices and fittings. C_{vi} are a measure of valve capacity – the flow rate of water through the fully opened valve when 1 Pa pressure drop is across the valve. $f_i(s_i)$ are the valve characteristics which indicate how the flow changes with valve stem position s_i. $0 \le s_i \le 1$. The standard relation for valve flow rate is:

$$\dot{Q} = C_v f(s)\sqrt{\Delta P/(\xi \cdot G)}$$

where ξ = 1 Pa and $G = \rho_{\text{process fluid}}/\rho_{H_2O\ @\ 70^\circ F}$. The quadratic relation, $f = s^2$ is a reasonable representation of a modified equal percentage trim. This is one reason why the flow rate is a nonlinear response to controller output.

Valves are air operated, and respond approximately in a first-order manner to controller output changes. Valve time-constants are τ_{vi}.

If either $\dot{m}_1, \dot{m}_2, T_{1in}$ or T_{2in} change, then the mixed fluid temperature, at the mixing point, immediately changes. However, the TT3 won't "feel" that new temperature until after the transport delay through length L_T. Even so, the TT3 response will not be immediate because of heat transport into the thermowell device. (Consider a thermometer's transient response when you take your temperature.) The sensor dynamics are modeled as third-order with time-constants τ_{T1}, τ_{T2}, and $\tau_{T3.}$

For convenience, fluid properties are assumed independent of temperature and pressure at the values indicated.

∈ is the piping internal surface roughness, and the ∈/D ratio is used for friction factor calculations. At the nominal conditions Re_1 = 10, 610, Re_2 = 5, 305, and Re_3 = 15, 915; and the fanning friction factors are f_{f1} = 0.0075, f_{f2} = 0.0090, f_{f3} = 0.0068. For convenience, the simulator uses these constant values, independent of actual flow rates.

The simulator determines how FT1, FT2, FT3, and TT3 respond in time to P_1, P_2, P_3, Tin_1, Tin_2, and controller outputs, O_1 and O_2. Then noise and drift are added to model the sensor response to the true the process values – to create the measured values.

First, the simulator determines how stem positions s_1 and s_2 dynamically respond to O_1 and O_2 (s_1 and s_2 lag behind in a first-order manner). Then, it determines how the fluids accelerate, in time, toward their new values. Then, it delays the mixed temperature by $\theta = L_T/v_3$, then third-order lags the delayed temperature to obtain TT3. Flow rate sensor dynamics are presumed instantaneous. Instead of a simple Euler's first order explicit method to numerically solve for the transients, because the flow dynamics are much faster than the valves or thermowell responses, a simple Predictor-Corrector numerical method is used to calculate all transient responses. I have seen this called a second-order Runge-Kutta method.

If toggled "ON", the simulator adds drifts to the pressure driving forces, drifts to the inlet temperatures, drifts to the friction loss coefficients, "stick-tion" to the valves, bias to all measurements, pressure spikes to upset flow rates, and noise to all flow measurements. Set enviro=1 to create the confounding effects, and set enviro=0 to turn them off.

The user controls events and their schedule in the subroutine "Events".

The simulator presents a strip-chart display on the computer screen so that you can observe transients.

The valve stem ODE's are

$$\tau_v \frac{ds_i}{dt} + s_i = \frac{p_i - 3}{15 - 3}$$

Where p_i represents the instrument air pressure on the actuator. Ideally, with perfect device calibrations, p_i is a 3-15 psig response to the electrical current input to the i/p transducer, and the current is a 4 to 20 mA response to the 0-100% controller output. You could adjust the 3, 15, 4, and 20 coefficients in the program to represent a non-ideal calibration situation.

The temperature ODE's are

$$\tau_{Ti} \frac{dT_{fi}}{dt} + T_{fi} = T_{fi}\ \text{target}$$

Where

$$T_{f1}\ \text{target} = \frac{T_{1in}\ \dot{m}_1 + T_{2in}\ \dot{m}_2}{\dot{m}_1 + \dot{m}_2}$$

$$T_{f2}\ \text{target} = T_{f1}$$

$$T_{f3}\ \text{target} = T_{f2}$$

The fluid flow ODE's start with a momentum balance on the fluid.

$$F = \frac{m\ a}{g_c} = \frac{\pi R^2 L \rho}{g_c} \frac{dv}{dt} = \frac{L}{g_c} \frac{d\dot{m}}{dt}$$

Where F is the sum of all forces trying to accelerate, or decelerate, the fluid.

$$F = \Delta P_{\underset{driving}{Pressure}} \cdot \pi R^2 + \Delta P_{head}\ \pi R^2 - \Delta P_{friction}\ \pi R^2 - \Delta P_{valve}\ \pi R^2$$

$$\Delta P_{pi} = P_{in} - P_3$$

$$\Delta P_{hi} = \rho g\left(h_i - h_3\right)/g_c$$

$$\Delta P_{fi} = 4 f_f\ \frac{Lequ}{D}\ \frac{1}{2}\ \rho v_i^2 / g_c = \frac{f_f\ Lequ\ \dot{m}_i^2}{\pi^2 R^5 \rho g_c}$$

Using

$$\dot{m}_i = \rho \dot{Q}_i = \rho C_v f_i\left(s_i\right)\sqrt{\Delta P_v / \xi G}$$

$$\Delta P_v = \frac{\dot{m}_i^2\ \xi G}{\rho^2 C_v^2 f_i^2\left(s_i\right)}$$

Combining, the ODE for the fluid dynamics becomes,

$$\frac{d\dot{m}_i}{dt} = \frac{g_c \pi R^2}{\left(L_i + L_3\right)}\ \Delta P_{Pi} + \frac{\pi R^2 \rho g}{\left(L_i + L_3\right)}\ \Delta h_i - \frac{f_{fi}\ Lequ_i}{\pi R^3 \rho\left(L_i + L_3\right)}\ \dot{m}_i^2 - \frac{g_c \xi \pi R^2}{\left(L_i + L_3\right)\rho_i \rho_w C_{vi}^2}\ \frac{\dot{m}_i^2}{f_i^2\left(s_i\right)} - \frac{f_{f3}\ Lequ_3}{\pi R^3\left(L_i + L_3\right)\rho}\left(\dot{m}_i + \dot{m}_2\right)^2$$

which can be simplified by lumping constants

$$\frac{d\dot{m}_i}{dt} = a_i \Delta P_i + b_i \Delta h_i - c_{i1}\dot{m}_i^2 - c_{i2}\left(\dot{m}_1 + \dot{m}_2\right)^2 - d_i \frac{\dot{m}_i^2}{f_i^2\left(s_i\right)}$$

Given the system dimensions and fluid properties

$a_1 = 0.3016$	$\frac{kg/\min}{s}/kPa$
$b_1 = 2.9576$	$\frac{kg/\min}{s}/m$
$c_{11} = 0.003979$	$\frac{kg/\min}{s}/(kg/\min)^2$
$c_{12} = 0.01082$	$\frac{kg/\min}{s}/(kg/\min)^2$
$d_1 = 0.002327$	$\frac{kg/\min}{s}/(kg/\min)^2$
$a_2 = 0.3427$	$\frac{kg/\min}{s}/kPa$
$b_2 = 3.3609$	$\frac{kg/\min}{s}/m$

c_{21} = 0.008139 $\frac{\text{kg/min}}{\text{s}} / (\text{kg/min})^2$

c_{22} = 0.01230 $\frac{\text{kg/min}}{\text{s}} / (\text{kg/min})^2$

d_2 = 0.01058 $\frac{\text{kg/min}}{\text{s}} / (\text{kg/min})^2$

Where $\dot{m}$ is kg/min and P is kPa.

As a note, the valve and upstream pressure requirements were determined as follows. Using nominal conditions $\dot{m}_1 = 20 \text{ kg/min}$ $\dot{m}_2 = 10 \text{ kg/min}$

Calculate ΔP_f

$\text{Re}_1 = 10610$ $\text{Re}_2 = 5305$ $\text{Re}_3 = 15915$

$f_{f1} = 0.0075$ $f_{f2} = 0.0090$ $f_{f3} = 0.0068$

$\Delta P_{f1} = 5.3\text{kPa}$ $\Delta P_{f2} = 2.4\text{kPa}$ $\Delta P_{f3}^2 = 32.3\text{kPa}$

$\Delta P_{f1+3} = 37.6 \text{ kPa}$ $\Delta P_{f2+3} = 34.7 \text{ kPa}$

Choose $\Delta P_v = \frac{1}{3} \Delta P_f$ and $f(s) = .5$

$C_{v1} = 6 \text{ x } 10^{-6} \text{ m}^2/\text{s} @ 1 \text{ Pa} = 7.9 \text{ gpm} @ 1 \text{ psi}$

$C_{v2} = 3 \text{ x } 10^{-6} \text{ m}^2/\text{s} @ 1 \text{ Pa} = 3.9 \text{ gpm} @ 1 \text{ psi}$

Then calculate $P_{\underset{\text{required}}{up}} = P_3 + \Delta P_f + \Delta P_v - \Delta P_h$

$P_1 = 180 \text{ kPa}$

$P_2 = 210 \text{ kPa}$

The simulator code is written in VBA for Excel.

The transport delay is handled by storing the mixed fluid temperature at the mixing point in an array, then sampling back in the array an appropriate number of times (nt=delay/dt=Lv/dt) for the temperature of the fluid that is influencing the T3 measurement. This works perfectly when the velocity is at a steady state, and it works well as long as the velocity does not change too fast relative to the time interval. With dt for the process simulator of 0.01 sec, and valve time-constants of 1 sec or greater the delay mechanism never jumps more than 1 element per time step. The array contains 2000 elements, permitting adequate storage for a slow velocity, with both valves nearly closed, that creates a transport delay of 20 sec. Rather than incrementing elements in the array at each time interval, the simulator uses a “clock” model in which the write and read pointers are incremented at each sampling.

Appendix B – Heuristic Cyclic, Direct Search, Multi-Variable Optimizer

"Direct Search" uses function evaluation and constraint violation only. No derivative or second derivative (no gradient, no Hessian).

"Cyclic" Cycle through each DV, making an incremental change in each, one at a time, then repeat the cycle. Each complete cycle marks a stage, or iteration

"Heuristic" Human logic rules guide the search increment expansion, reversal and contraction.

The procedure is:

1. Initialize DV base (the initial trial solution in a feasible region) and evaluate the OF (Objective Function, measure of goodness)
2. Start the cycle for one iteration (sometimes called epoch or stage)
3. Taking each DV, one at a time, set the new trial DV value,
$$DV_{trial} = DV_{base} + \Delta DV_i$$
Note DV_{trial} and DV_{base} are vectors. Only add ΔDV_i to the ith element.
4. Evaluate the function at the trial value. If worse or "FAIL" keep $DV\ base$ and set ΔDV_i to a smaller change in the opposite direction
$$\Delta DV_i = -contract * \Delta DV_i$$
5. Otherwise, keep the better solution (make it the base point) and accelerate moves in the right direction
$$OF_{base} = OF_{trial}$$
$$DV_{base} = DV_{trial}$$
$$\Delta DV_i = expand * \Delta DV_i$$
6. At the end of the cycle, check for stopping criteria (excessive iterations, or meet convergence criteria).
7. Stop, or repeat from Step 2

This algorithm is simple to understand and simple to implement.

Direct searches can accommodate hard constraints on either the DV or auxiliary variables.

Direct searches do not have $\underline{\nabla}$ or $\underline{\underline{\nabla}}^2$ operators that 1) have difficulty coping with non-analytic surfaces (ridges, discontinuities in function or slope, noise or variability on an OF value), and 2) add computational burden and programming complexity.

Although other search algorithms may be more efficient (use fewer iterations, or fewer function evaluations), to my taste, the simplicity, robustness, and constraint accommodation of this algorithm are substantial advantages.

Appendix C – Leapfrogging, Direct Search, Multi-Player, Multi-Variable Optimizer