



Information Systems Laboratories, Inc.

PWR Steady-State Topics and Exercises

Information Systems Laboratories, Inc.

TRACE/SNAP User Workshop



Objectives

The objective of this morning's discussions and exercises is to provide insights in obtaining steady-state conditions using the PWR power plant model. At the end of this morning, the PWR model will be ready to run a cold leg small-break LOCA.



Morning Session Topics

- **Managing Multiple Simulations in a Single Model**
- Achieving Steady-State Target Conditions
- Debugging TRACE Input Errors
- Break Modeling & Validation
- Reflood Configuration and Steady-State Calculation



One Model – Multiple Simulations

Multiple accident simulations for a nuclear plant, such as an LBLOCA and an SBLOCA, may be performed using a TRACE model. Accidents may assume different initial and boundary conditions (e.g., power level, ECCS flow assumptions, etc.)

Maintaining multiple models is inefficient when model updates are required, and can lead to inconsistencies.



One Model – Multiple Simulations

How do you conveniently maintain multiple accident configurations within a single model?

- Control Systems*
- Restart Input File*
- SNAP Numerics

* Discussed only briefly



One Model – Multiple Simulations

Control Systems can be included in a model to manage differences between simulations:

Advantages:

- ⊕ Single Base TRACE Input File
- ⊕ Minimal model reconfiguration via restart file (if well organized)

Disadvantages:

- ⊖ Limited Scope - Some properties are not controllable
- ⊖ Less Maintainable – Can lead to complex control systems. SNAP layout/annotation can help significantly.



One Model – Multiple Simulations

Restart files can be used to manage differences between simulations:

Advantages:

- ⊕ Flexibility - Most components can be updated via restart files
- ⊕ Simple Reconfiguration – When combined appropriately with control systems
- ⊕ Model changes are documented (They are explicit in the restart file)

Disadvantages:

- ⊖ Complexity – Restart files can become large and complex if components are modified
- ⊖ Less Maintainable – Corrections in base model may be unintentionally overwritten in restart deck
- ⊖ Fragile – Changes in base deck can cause restart to be incompatible
- ⊖ Simulation differences are not incorporated in the base model



One Model – Multiple Simulations

SNAP Numerics is a more recent option that can be used to manage accident simulation differences:

Advantages:

- ⊕ Flexibility – Most model parameters are configurable via Numerics
- ⊕ Maintainability – Differences are included in the base SNAP model
- ⊕ Robustness – Changes to model do not typically impact Numerics
- ⊕ Simplicity – Can simplify implementation for some configuration changes
- ⊕ Differences are included in base SNAP model

Disadvantages:

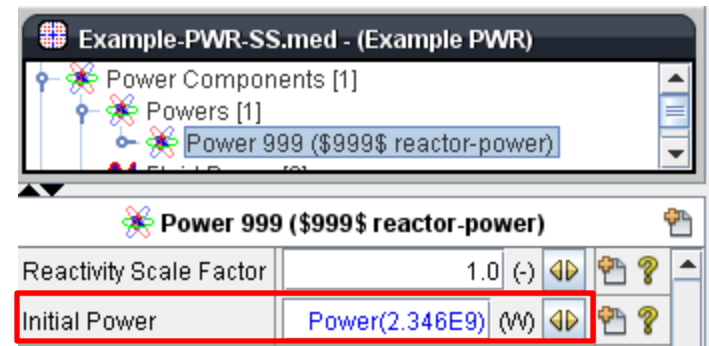
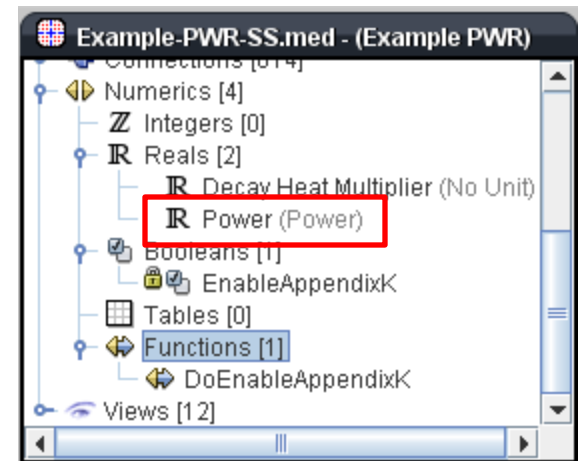
- ⊖ One SNAP Model, but multiple TRACE input files
- ⊖ There is not a way from within SNAP to see where Numerics is used in the model. (However, you can export an input file with Numerics preserved, which shows where Numerics variables is used in the model.)



SNAP Numerics

What is SNAP Numerics?

- SNAP Variables – Used to configure model parameters
- Programming tools used to configure these variables:
 - Python (Built in)
 - Matlab
 - Mathcad





Python

- Python is a popular scripting language
- You don't need to know much Python to be effective in SNAP!
- There are many simple tutorials available on the web.

Brief Python Basics

- # is used before comments
- Indentation is used to delineate code blocks (such as for 'if')

```
# Get variable that tells whether Appendix K
# Options are used.
isAppendixK = GetVariable("EnableAppendixK")

decay_heat_multiplier = 1.0
power = 2.3E9

if isAppendixK == True:
    decay_heat_multiplier = 1.2
    power = power*1.02

SetVariable("Decay Heat Multiplier", decay_heat_multiplier)
SetVariable("Power", power)
```



Numerics Exercise

The PWR model contains fictional best estimate conditions. Configure the model via Numerics to use power that is 102% of best estimate conditions (an Appendix K assumption).

Refer to the Day4 Morning Exercise: **PWR Model – Numerics Exercise** located in the Day4 morning section of your workbook or [NumericsExercise.pdf](#) located in the [Day4\Morning\PWR1_Numerics](#) folder.







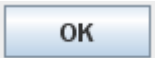
Storing Multiple Steady-States

SNAP allows multiple sets of steady-state conditions to be stored in the SNAP model. This is useful when separate accident simulations begin from different steady-state conditions.



Saving SS Conditions Exercise

The initial conditions in the model are currently best estimate conditions. Save the best estimate initial conditions by doing the following (Appendix K initial conditions will be save at a later point):

1. In the model click on  Model Options
2. In the properties window click  on 
3. In the 'Manage Initial Conditions' dialog click 
4. On the item that appears in the initial conditions list, double click on 'unnamed'. Replace 'unnamed' with 'Best Estimate' and click 

This makes it possible to restore best estimate conditions even after other initial conditions have been imported into the model.



Morning Session Topics

- Managing Multiple Simulations in a Single Model
- **Achieving Steady-State Target Conditions**
- Debugging TRACE Input Errors
- Break Modeling & Validation
- Reflood Configuration and Steady-State Calculation



Steady-State Challenge

In achieving steady-state conditions, there are multiple target conditions that need to be achieved simultaneously.

(e.g., Loop Flows, Primary Side Pressure, Tave, etc.)

Parameters changed to achieve one target condition often affect other target conditions.



Steady-State Challenge

How do you efficiently achieve all steady-state target conditions?

Manually adjusting parameters to achieve steady-state is a time consuming process that has to be repeated each time targets change. Typically this requires several iterations.

Control Systems can be used to adjust control parameters automatically in order to achieve steady-state target values. Targets can typically be achieved in a single steady-state run.



Steady-State Challenge

How do you efficiently achieve all steady-state target conditions?

- Manual Adjustment
(Some parameters are not controllable)
- Constrained Steady-State (CSS)
(Built in Control Systems)
- User Defined Control Systems



Manual Adjustment

Some parameters are not adjustable via control systems and must be adjusted manually.

For example, K-losses may need to be adjusted in order for pressure losses to match plant data. K-losses are not a controllable parameter in TRACE.

It is recommended that manual adjustments be made after control systems have been implemented for other steady-state targets.



Constrained Steady-State (CSS)

TRACE has built-in steady-state control systems that can be enabled for a few common cases. These include:

- CSS Type 1 - Controls **Pump speed** to achieve a target **mass flow** or **velocity**
- CSS Type 2 - Controls **Valve area** to achieve a **mass flow** or **upstream pressure**
- ⊕ Simple to add – 1.) Specify Component and 2.) Choose Target Value.
- ⊕ Achieves steady-state fairly rapidly
- ⊖ Limited to a few cases
- ⊖ Not usable during transient



CSS Exercise

Add a constrained steady state controller to PUMP 315

- Set the target mass flow to 4259 kg/s
- Set the max pump speed to 200 rad/s

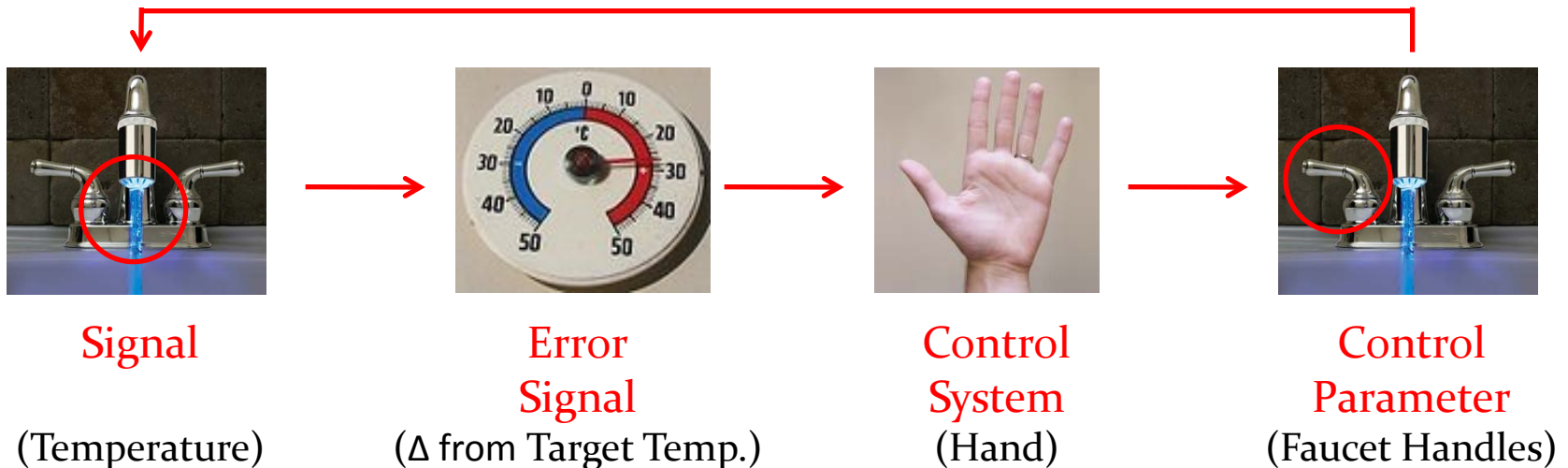
(See CSS_Exercise1.pdf)

Refer to the Day4 Morning Exercise: **Constrained Steady-State Exercise** located in the Day4 morning section of your workbook or [CSS_Exercise1.pdf](#) located in the [Day4\Morning\PWR2_Achieving_Steady-State](#) folder.



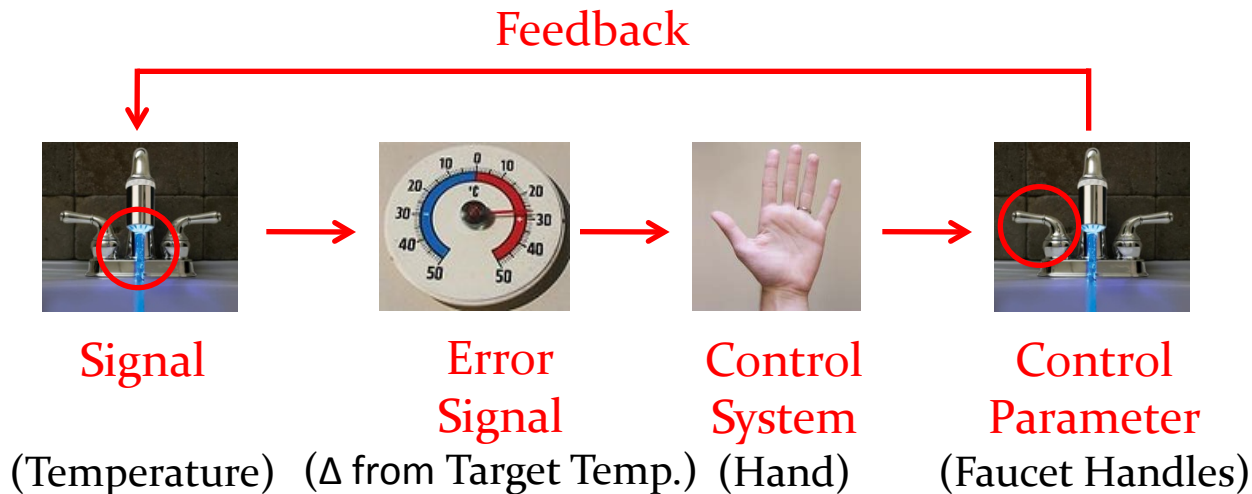
User Defined Control Systems

A typical control system is comprised of a **Signal**, an **Error Signal**, a **Control System**, and a **Control Parameter**. The control system adjusts the control parameter to minimize the error. For simple control systems, there should be a monotonic relationship between the control parameter and the error signal (a consistently positive or negative slope relationship).



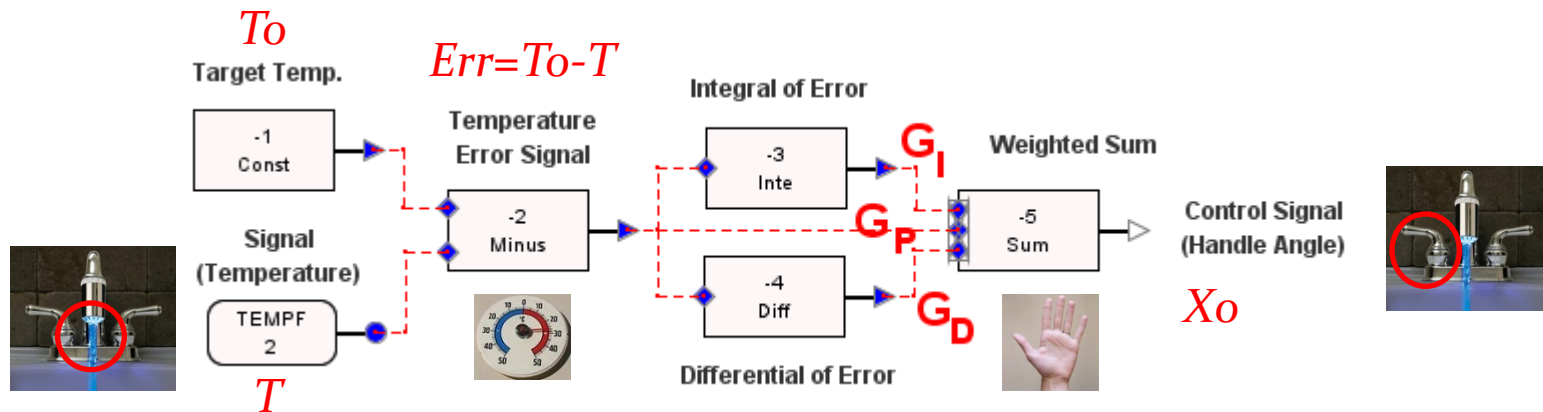
Control System Stability

The control system acts on a feedback loop. When the control parameter changes, there is a delay time before the error signal measures the response (e.g., the time for the water to go from the valve to the faucet exit). If the control parameter is adjusted too quickly, it can overcorrect. If it overcorrects significantly, the error signal can amplify and become unstable (think feedback through a microphone). If the parameter is adjusted too slowly, the time to reach steady state can be excessive.



Control System Theory

A common controller type is the **PID controller**. The control signal value is set **(P)**roportional to the Error, the **(I)**ntegral of the Error, and the **(D)**ifferential of the Error. One of the challenges is choosing the P, I, & D Gains to optimize the time to reach steady-state. If the Differential gain G_D is set to zero, this is called a **PI controller**. If Integral gain G_I is also set to zero, this is called a **P controller**.



$$X_o = G_P Err + G_I \int Err \cdot dt + G_D \frac{d(Err)}{dt}$$



Gain Guidelines

The G_P , G_I , and G_D gains should all be positive or negative. The sign of the gains depends on whether there is a positive or a negative slope correlation between the control variable X_o and error signal Err .

- A **positive slope** correlation indicates **negative gains** are needed, and a **negative slope** correlation indicates **positive gains** are needed.
- If the sign of the gains is chosen incorrectly, then the error will typically diverge from zero without exhibiting persistent oscillations about the target value.
- If the error signal shows persistent oscillations about the target value, this suggests that the sign on the gains is correct, but the gain is too large causing overshoot (i.e. overcorrection).



Control System Theory

The control system needs to supply a Control Signal value (e.g., Faucet Handle Position), which defines the form of the control logic. However, we are interested in the responsiveness and effectiveness of the controller, so the derivative of the control system equation is more convenient to examine.

$$\text{P Controller: } \frac{d(Xo)}{dt} = G_P \frac{d(Err)}{dt}$$

$$\text{PI Controller: } \frac{d(Xo)}{dt} = G_P \frac{d(Err)}{dt} + G_I Err$$

$$\text{PID Controller: } \frac{d(Xo)}{dt} = G_P \frac{d(Err)}{dt} + G_I Err + G_D \frac{d^2(Err)}{dt^2}$$

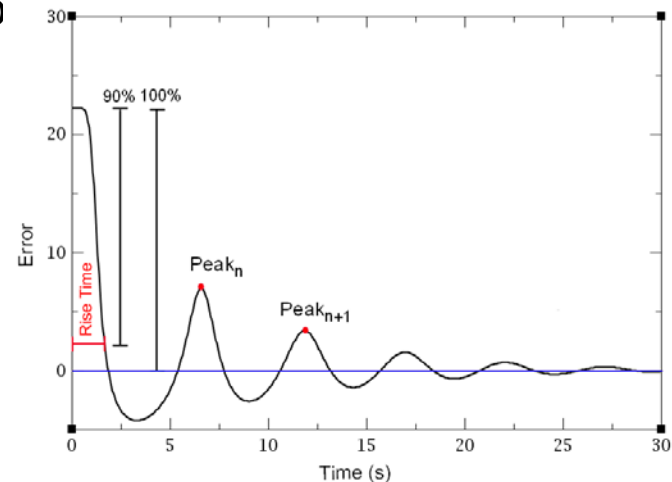
Note that the Proportional controller reaches equilibrium when the **Change in Error** (not the Error) goes to zero. Proportional controllers often settle at a constant error. Thus proportional controllers are often a poor choice. **The integral term is necessary to make the error go to zero.**



PID Characterization

Four major parameters are used to characterize a PID controller:

1. **Rise Time** – The initial time it takes for the Error to be reduced by 90%.
2. **Overshoot** – $|\text{Error Peak } (n+1)|/|\text{Error Peak } (n)|$ **For oscillatory error**
3. **Settling Time** – Time it takes for Error to
4. **Steady-State Error Offset**





PID Tuning

Assuming the sign of the gains is correct, the typical effect of increasing $|G_P|$, $|G_I|$, and $|G_D|$ is summarized in the Table below:

	Rise Time	Overshoot	Settling Time	S-S Error
$ G_P $	Reduce	Increase*	-	Increase
$ G_I $	Reduce	Increase*	Reduce /(Inc.*)	Remove
$ G_D $	-	Reduce*	Reduce	-

Tips:

*When overcorrection occurs signaled by oscillations

- Use non-zero G_I to achieve the correct target.
- Increase G_P or G_I if convergence is slow and there is no overshoot
- Decrease G_P or G_I if oscillations are noticeable & damping is slow
- Add G_D to improved Settling
(Warning - Differential term will amplify noise in the Error signal)

PID Configuration

The Ziegler-Nichols method is a simple structured tuning technique for PID controllers. The Steps are:

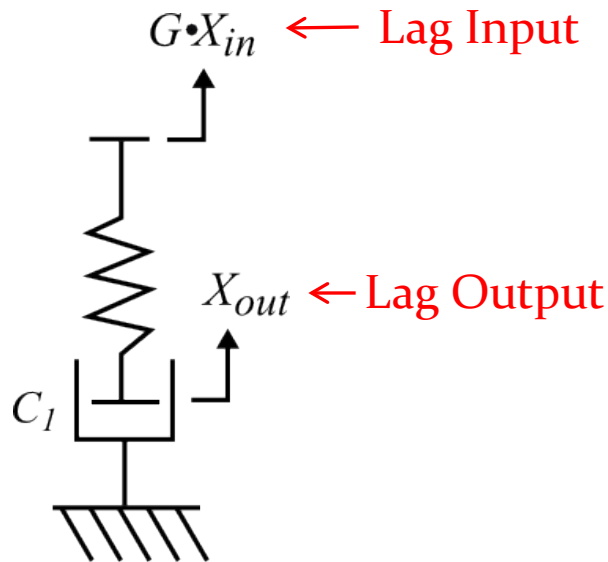
1. Set the Integral and Differential gains (G_I and G_D) to zero
2. Tune the Proportional gain (G_P) till the Error oscillates with constant amplitude (Borderline unstable) – This will be called G_{PMax}
3. From the Error response take the oscillation Period T_P (Peak to Peak time)
4. Depending on the type of controller desired, modify the gains based on the following Table:

	G_P	G_I	G_D
P Controller	$0.5 G_{PMax}$		
PI Controller	$0.45 G_{PMax}$	$1.2 G_{PMax}/T_P$	
PID Controller	$0.6 G_{PMax}$	$2 G_{PMax}/T_P$	$T_P G_{PMax}/8$



Reducing Noise

When the input signal has a significant amount of noise it is often desirable to filter out the noise. A lag Control Block [26] can be added to the PID controller output to filter noise effects. The lag control acts like a spring-damper system. CBCON1 (C_1) is the lag (or damping) constant.



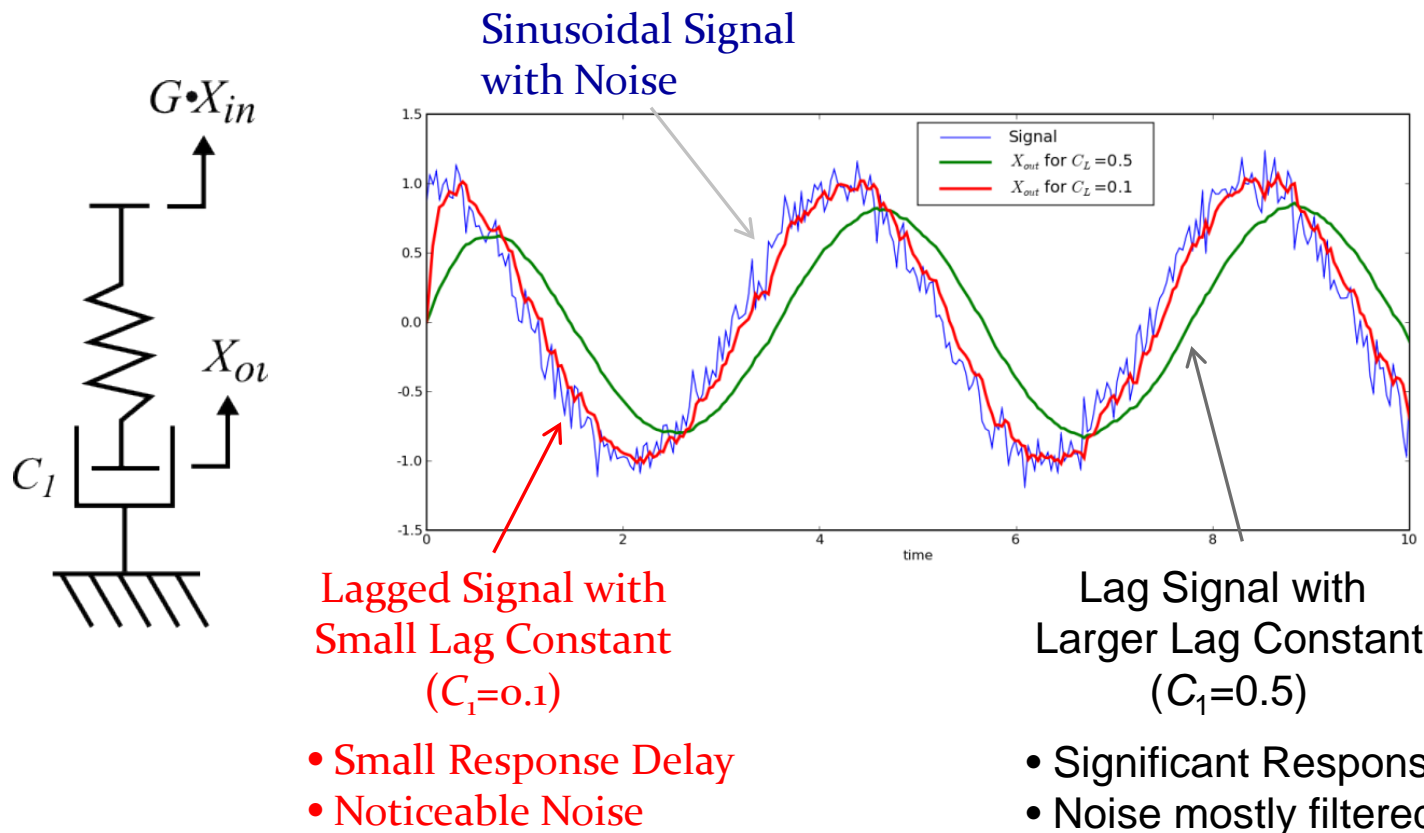
Lag Control Equation:

$$\underbrace{C_1 \cdot \frac{dX_{out}}{dt} + X_{out}}_{\text{Spring-Damper Terms}} = \underbrace{G \cdot X_{in}}_{\text{Forcing Term}}$$

Spring-Damper Terms

Reducing Noise

Increasing the lag constant C_1 increases the filtering of the high frequency noise, but reduces the responsiveness of the control signal.





PID with Lag Tuning

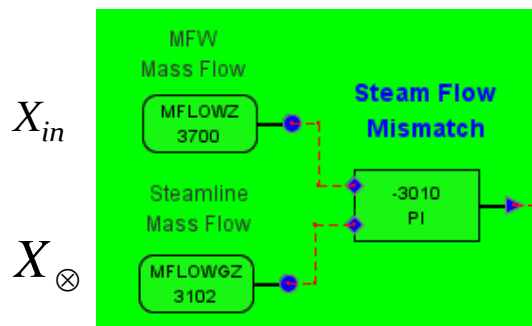
Assuming the sign of the gains is correct, and that a lag control of Gain 1 and Lag constant C_L is added to the PID output, the typical effect of increasing $|G_P|$, $|G_I|$, $|G_D|$, and C_L is summarized in the Table below:

	Rise Time	Overshoot	Settling Time	S-S Error	Noise
$ G_P $	Reduce	Increase*	-	Increase	-
$ G_I $	Reduce	Increase*	Reduce or Increase*	Remove	-
$ G_D $	-	Reduce*	Reduce	-	Increase
C_L	Increase	Reduce*	Increase or Reduce*	-	Reduce

*When overcorrection occurs signaled by oscillations

TRACE PI and PID Controls

For convenience, TRACE includes PI and PID control blocks. The PI and PID controls include a lag block. The PI control is characterized below:



$$\text{Target Value} = X_{\otimes} \quad \text{Err} = X_{\otimes} - X_{in}$$

$$C_L \cdot \frac{dX_{out}}{dt} + X_{out} = G \left(\text{Err} + \frac{1}{T} \int \text{Err} \cdot dt \right)$$

X_o is the initial value of X_{out}

* The target X_{\otimes} can be specified via a control block (ICB2). Otherwise it is defined via PI control constant 1 (CBCON1)

G	Type	[200] PI controller		
	Description	<none>		
G	Gain	1.0	(-)	⏮ ⏭ ⏰
	Minimum	-1.0	(-)	⏮ ⏭ ⏰
	Maximum	1.0	(-)	⏮ ⏭ ⏰
X_{\otimes}	Constant 1	0.0	(-)	⏮ ⏭ ⏰
	Constant 2	0.049264191	(-)	⏮ ⏭ ⏰
T	Delta T	100.0	(s)	⏮ ⏭ ⏰
C_L	Time Constant	2.0	(s)	⏮ ⏭ ⏰

Conversion from G_P & G_I to TRACE PI constants G & T

$$G = G_P \quad T = \frac{G_P}{G_I}$$



Control Systems Exercise

In the PWR model, tune the Tave controller using the Ziegler-Nichols method. The parameter $G_{PM_{\max}}$ has already been set as a starting point.

Refer to the Day4 Morning Exercise: **PID Controller Exercise** located in the Day4 morning section of your workbook or [PIDExercise2.pdf](#) located in the [Day4\Morning\PWR2_Achieving_Steady-State](#) folder.

Morning Session Topics

- Managing Multiple Simulations in a Single Model
- Achieving Steady-State Target Conditions
- **Debugging TRACE Input Errors**
- Break Modeling & Validation
- Reflood Configuration and Steady-State Calculation

Debugging Input Errors

TRACE input is complex, which increases the potential for input errors.

The error messages printed by TRACE are not always informative.

How do you identify the location of an input error so that it can be fixed?



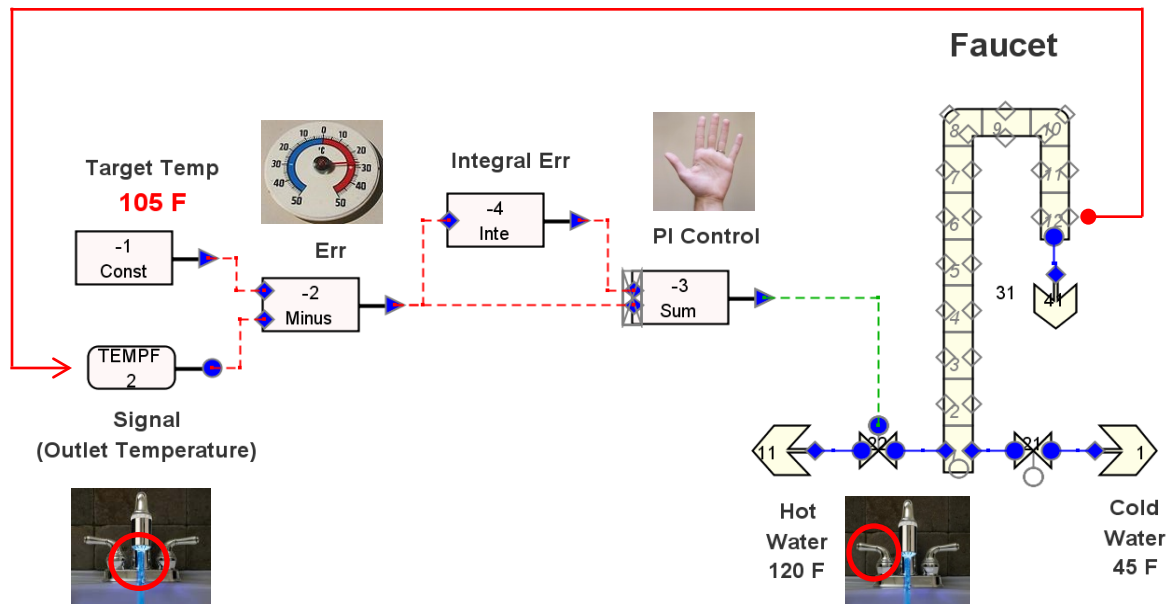
Debugging Input Errors

Suggested Steps:

1. Review ALL errors and warnings carefully
(The relevant error is often reported before the 'fatal error' message)
2. Look for the string 'error' in the *.echo file
(The error file may report the line where an error occurred)
3. Look at where the *.out file stopped
(Sometimes stops at or near the error location – look at what component was being processed when *.out file stops)
4. Compare input file against recently working model
(Use of a source control is recommended for evolving models)

Input Debugging Exercise - Water Faucet Model

A simple faucet model with some errors will be used as a debugging exercise. A PI controller is used for the hot water valve such that the temperature should reach 105 °F (313.7 K). Correct the input errors that cause the model to fail and verify that the controller causes the faucet to reach 105 °F. There are 3 errors, but one will not be reported till the first 2 are fixed.



Input Debugging Exercise

Refer to the Day4 Morning Exercise: **Debugging TRACE Input Errors** located in the Day4 morning section of your workbook or [TRACE_Input_Error_Debugging.pdf](#) located in the [Day4\Morning\PWR3_Input_Debugging](#) folder.



Morning Session Topics

- Managing Multiple Simulations in a Single Model
- Achieving Steady-State Target Conditions
- Debugging TRACE Input Errors
- **Break Modeling & Validation**
- Reflood Configuration and Steady-State Calculation



Modeling Breaks via Valves

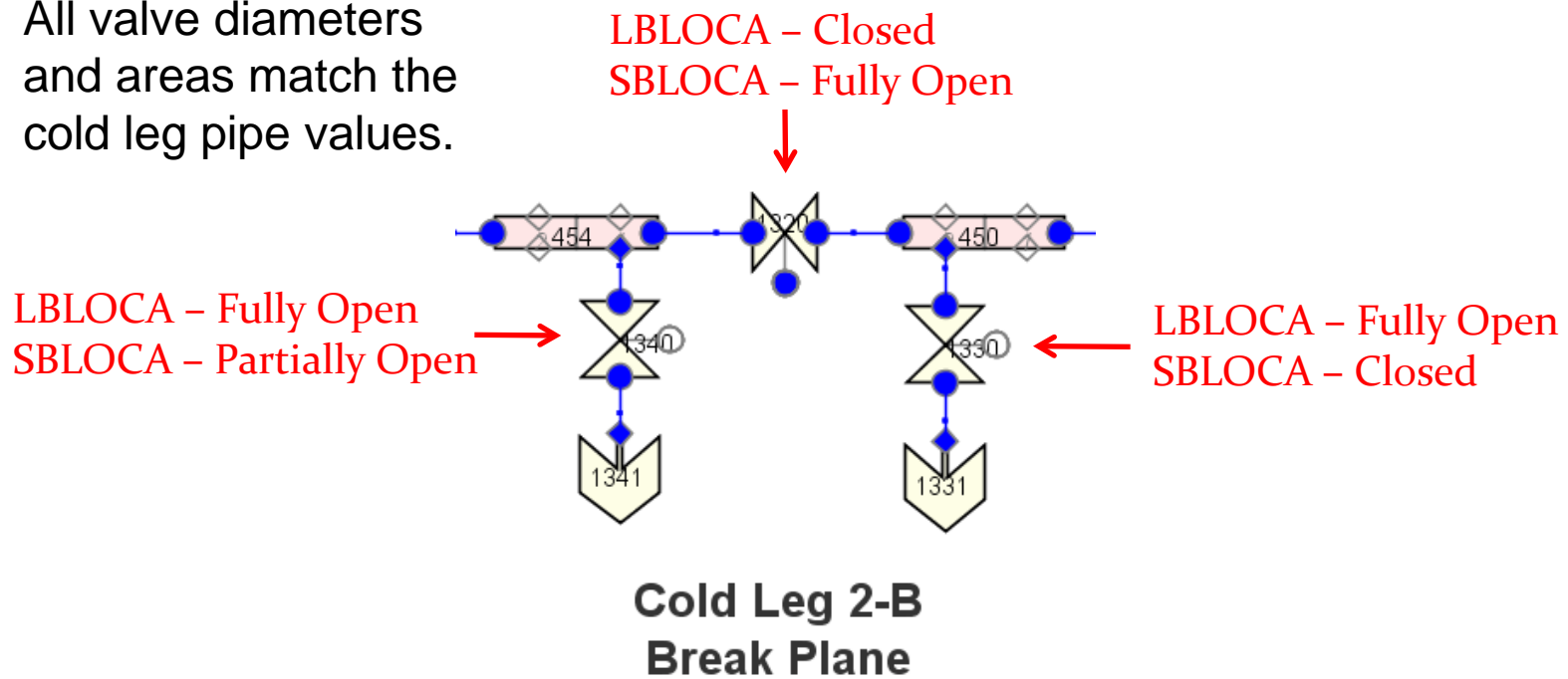
Valve components are a convenient way to model breaks. Some of the advantages are:

- Valves are easily configured to open in response to trips. Break assumptions are explicitly included in the model when the break valve opens in response to an accident specific Trip (e.g., an SBLOCA trip). This leads to better self-documentation for the model.
- Valves can be resized via control systems. Break size logic can be included in the control systems, so a single valve may be configured to handle multiple break sizes. For example, a single valve can participate in both an LBLOCA and an SBLOCA simulation with break sizes managed in the control system.

Modeling Breaks via Valves

Below is a typical combination double guillotine LBLOCA and SBLOCA cold leg break model. The valves can be configured to behave as follows:

All valve diameters and areas match the cold leg pipe values.



Valve Caveat

By default, valves include a **built in form loss** which impacts break flow. In the current version of TRACE, valves include an option to turn off this internal form loss and instead specify an explicit flow area fraction vs. K Loss table:

Valve Internal Loss Parameter

INTLOSSOFF	0 – Internal Loss Enabled
	1 – Internal Loss Disabled (Include 'kopen' Table)



Containment Break Modeling

For the break component, the effect of the length (DXIN) and volume (VOLIN) were covered on Day 3 in the afternoon. The conclusion was that for a large volume pressure sink $VOLIN/DXIN$ should be large (i.e. the Area should be large), so that the pressure of the break represents a stagnant pressure.

Containment Break Modeling

Break pressure is of course important since this governs the break flow and the timing of choked flow events. Sometimes backflow from containment occurs, so break mixture conditions are also important. To avoid liquid entering via backflow through the break do the following:

- Set initial gas volume fraction (ALPIN) to 1.0

And do one of the following

- Set temperature table option (ISAT) to '[3] Set liquid and gas to Tsat' to cause saturated steam to enter during backflow.

OR

- Set the noncondensable partial pressure (PAIN) close to the initial pressure and set the initial mixture temperature (TIN) to containment conditions.

Break Modeling Exercise

Add a 3 inch cold leg break to the Loop 1 cold leg PWR model using the directions provided in the 'Break Modeling Exercise' document.

Refer to the Day4 Morning Exercise: **Break Modeling Exercise** located in the Day4 morning section of your workbook or [BreakModelingExercise.pdf](#) located in the [Day4\Morning\PWR4_Break_Model](#) folder.



Morning Session Topics

- Managing Multiple Simulations in a Single Model
- Achieving Steady-State Target Conditions
- Debugging TRACE Input Errors
- Break Modeling & Validation
- **Reflood Configuration and Steady-State Calculation**



Reflood Configuration

The example PWR model has been set up to perform a small break loss-of-coolant-accident. It is expected that during this transient, core uncovering will occur and the fuel rods will heat up. Emergency core cooling systems are expected to activate, injecting coolant into the primary system. Core reflood is expected to occur, stop the fuel rod heatup process and subsequently quench the fuel rods.



Reflood Configuration Exercise

To predict a more accurate representation of the fuel rod behavior during the transient, the reflood and fine-mesh renodalization options should be activated in the model. Check the PWR model for reflood and fine-mesh capabilities. Make the appropriate modifications to activate the reflood and fine-mesh capabilities if needed.

Refer to the Day4 Morning Exercise: **Reflood Activation Exercise** located in the Day4 morning section of your workbook or [Reflood_Activation_Exercise.pdf](#) located in the **Day4\Morning\PWR5_Steady-State** folder.



PWR Steady-State Calculation Exercise

The model is now ready to run a steady-state calculation. The small break LOCA transient calculation will restart from the end of the steady-state run.

Run a 1000 s steady-state calculation.

Refer to the Day4 Morning Exercise: **PWR Steady-State Calculation Exercise** located in the Day4 morning section of your workbook or [Steady-StateRunExercise.pdf](#) located in the [Day4\Morning\PWR5_Steady-State](#) folder.