Control Theory and Engineering

(Properties, Modeling and Applications)

Filiberto Loyd

First Edition, 2012

ISBN 978-81-323-3529-0

Published by:
University Publications
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter- 1

BIBO Stability

In electrical engineering, specifically signal processing and control theory, **BIBO stability** is a form of stability for linear signals and systems that take inputs. BIBO stands for *Bounded-Input Bounded-Output*. If a system is BIBO stable, then the output will be bounded for every input to the system that is bounded.

A signal is bounded if there is a finite value $B > 0$ such that the signal magnitude never exceeds B, that is

$$|h[n]| \leq B \quad \forall n \in \mathbb{Z} \text{ for discrete-time signals, or}$$
$$|h(t)| \leq B \quad \forall t \in \mathbb{R} \text{ for continuous-time signals.}$$

Time-domain condition for linear time invariant systems

Continuous-time necessary and sufficient condition

For a continuous time linear time invariant (LTI) system, the condition for BIBO stability is that the impulse response be absolutely integrable, i.e., its L^1 norm exist.

$$\int_{-\infty}^{\infty} |h(t)| \, \mathrm{d}t = \|h\|_1 < \infty$$

Discrete-time sufficient condition

For a discrete time LTI system, the condition for BIBO stability is that the impulse response be absolutely summable, i.e., its ℓ^1 norm exist.

$$\sum_{n=-\infty}^{\infty} |h[n]| = \|h\|_1 < \infty$$

Proof of sufficiency

Given a discrete time LTI system with impulse response $h[n]$ the relationship between the input $x[n]$ and the output $y[n]$ is

$$y[n] = h[n] * x[n]$$

where * denotes convolution. Then it follows by the definition of convolution

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k]$$

Let $\|x\|_\infty$ be the maximum value of $|x[n]|$, i.e., the supremum norm.

$$\begin{aligned}
|y[n]| &= \left| \sum_{k=-\infty}^{\infty} h[n-k]x[k] \right| \\
&\leq \sum_{k=-\infty}^{\infty} |h[n-k]|\,|x[k]| \qquad \text{(by the triangle inequality)} \\
&\leq \sum_{k=-\infty}^{\infty} |h[n-k]|\,\|x\|_\infty \\
&= \|x\|_\infty \sum_{k=-\infty}^{\infty} |h[n-k]| \\
&= \|x\|_\infty \sum_{k=-\infty}^{\infty} |h[k]|
\end{aligned}$$

If $h[n]$ is absolutely summable, then $\sum_{k=-\infty}^{\infty} |h[k]| = \|h\|_1 < \infty$ and

$$\|x\|_\infty \sum_{k=-\infty}^{\infty} |h[k]| = \|x\|_\infty \|h\|_1$$

So if $h[n]$ is absolutely summable and $|x[n]|$ is bounded, then $|y[n]|$ is bounded as well because $\|x\|_\infty \|h\|_1 < \infty$.

The proof for continuous-time follows the same arguments.

Frequency-domain condition for linear time invariant systems

Continuous-time signals

For a causal, rational, continuous-time system, the condition for stability is that the region of convergence (ROC) of the Laplace transform includes the imaginary axis. When the system is causal, the ROC is the open region to the right of a vertical line whose abscissa is the real part of the "largest pole", or the pole that has the greatest real part of any pole in the system. The real part of the largest pole defining the ROC is called the abscissa of convergence. Therefore, all poles of the system must be in the strict left half of the s-plane for BIBO stability.

This stability condition can be derived from the above time-domain condition as follows :

$$\begin{aligned}
&\int_{-\infty}^{\infty} |h(t)| \, \mathrm{d}t \\
&= \int_{-\infty}^{\infty} |h(t)| \left|e^{-j\omega t}\right| dt \\
&= \int_{-\infty}^{\infty} \left|h(t)(1 \cdot e)^{-j\omega t}\right| dt \\
&= \int_{-\infty}^{\infty} \left|h(t)(e^{\sigma + j\omega})^{-t}\right| dt \\
&= \int_{-\infty}^{\infty} \left|h(t)e^{-st}\right| dt
\end{aligned}$$

where $s = \sigma + j\omega$ and $\mathrm{Re}(s) = \sigma = 0$.

The region of convergence must therefore include the imaginary axis.

Discrete-time signals

For a causal, rational, discrete time system, the condition for stability is that the region of convergence (ROC) of the z-transform includes the unit circle. When the system is causal, the ROC is the open region outside a circle whose radius is the magnitude of the pole with largest magnitude. Therefore, all poles of the system must be inside the unit circle in the z-plane for BIBO stability.

This stability condition can be derived in a similar fashion to the continuous-time derivation:

$$\sum_{n=-\infty}^{\infty} |h[n]| = \sum_{n=-\infty}^{\infty} |h[n]| \left|e^{-j\omega n}\right|$$

$$= \sum_{n=-\infty}^{\infty} \left| h[n](1 \cdot e)^{-j\omega n} \right|$$
$$= \sum_{n=-\infty}^{\infty} \left| h[n](re^{j\omega})^{-n} \right|$$
$$= \sum_{n=-\infty} \left| h[n]z^{-n} \right|$$

where $z = re^{j\omega}$ and $r = |z| = 1$.

The region of convergence must therefore include the unit circle

Chapter- 2

Lyapunov Stability

In mathematics, the notion of **Lyapunov stability** occurs in the study of dynamical systems. In simple terms, if all solutions of the dynamical system that start out near an equilibrium point x_e stay near x_e forever, then x_e is **Lyapunov stable**. More strongly, if x_e is Lyapunov stable and all solutions that start out near x_e converge to x_e, then x_e is **asymptotically stable**. The notion of **exponential stability** guarantees a minimal rate of decay, i.e., an estimate of how quickly the solutions converge. The idea of Lyapunov stability can be extended to infinite-dimensional manifolds, where it is known as structural stability, which concerns the behavior of different but "nearby" solutions to differential equations. Input-to-state stability (ISS) applies Lyapunov notions to systems with inputs.

Definition for continuous-time systems

Consider an autonomous nonlinear dynamical system

$$\dot{x} = f(x(t)), \quad x(0) = x_0,$$

where $x(t) \in \mathcal{D} \subseteq \mathbb{R}^n$ denotes the system state vector, $\mathcal{D}$ an open set containing the origin, and $f : \mathcal{D} \to \mathbb{R}^n$ continuous on $\mathcal{D}$. Suppose f has an equilibrium x_e.

1. The equilibrium of the above system is said to be **Lyapunov stable**, if, for every ε > 0, there exists a δ = δ(ε) > 0 such that, if $\|x(0) - x_e\| < \delta$, then $\|x(t) - x_e\| < \epsilon$, for every $t \geq 0$.
2. The equilibrium of the above system is said to be **asymptotically stable** if it is Lyapunov stable and if there exists δ > 0 such that if $\|x(0) - x_e\| < \delta$, then $\lim_{t \to \infty} (x(t) - x_e) = 0$.

3. The equilibrium of the above system is said to be **exponentially stable** if it is asymptotically stable and if there exist α,β,δ > 0 such that if $\|x(0) - x_e\| < \delta$, then $\|x(t) - x_e\| \leq \alpha \|x(0) - x_e\| e^{-\beta t}$, for $t \geq 0$.

Conceptually, the meanings of the above terms are the following:

1. Lyapunov stability of an equilibrium means that solutions starting "close enough" to the equilibrium (within a distance δ from it) remain "close enough" forever (within a distance ε from it). Note that this must be true for *any* ε that one may want to choose.
2. Asymptotic stability means that solutions that start close enough not only remain close enough but also eventually converge to the equilibrium.
3. Exponential stability means that solutions not only converge, but in fact converge faster than or at least as fast as a particular known rate $\alpha \|x(0) - x_e\| e^{-\beta t}$.

The trajectory *x* is (locally) *attractive* if

$$\|y(t) - x(t)\| \to 0$$

for $t \to \infty$ for all trajectories that start close enough, and *globally attractive* if this property holds for all trajectories.

That is, if *x* belongs to the interior of its stable manifold. It is *asymptotically stable* if it is both attractive and stable. (There are counterexamples showing that attractivity does not imply asymptotic stability. Such examples are easy to create using homoclinic connections.)

Definition for discrete-time systems

The definition for discrete-time systems is almost identical to that for continuous-time systems. The definition below provides this, using an alternate language commonly used in more mathematical texts.

Let (*X*,*d*) be a metric space and $f: X \to X$ a continuous function. A point $x \in X$ is said to be **Lyapunov stable**, if, for each ε > 0, there is a δ > 0 such that for all $y \in X$, if

$d(x,y) < \delta$

then

$d(f^n(x),f^n(y)) < \varepsilon$

for all $n \in \mathbb{N}$.

We say that x is **asymptotically stable** if it belongs to the interior of its stable set, *i.e.* if there is a $\delta > 0$ such that

$$\lim_{n\to\infty} d(f^n(x), f^n(y)) = 0$$

whenever $d(x,y) < \delta$.

Lyapunov stability theorems

The general study of the stability of solutions of differential equations is known as stability theory. Lyapunov stability theorems gives only sufficient condition.

Lyapunov's second method for stability

Lyapunov, in his original 1892 work proposed two methods for demonstrating stability. The first method developed the solution in a series which was then proved convergent within limits. The second method, which is almost universally used nowadays, makes use of a *Lyapunov function V(x)* which has an analogy to the potential function of classical dynamics. It is introduced as follows. Consider a function $V(x) : R^n \to R$ such that

- $V(x) \geq 0$ with equality if and only if $x = 0$ (positive definite)
- $\dot{V}(x) = \frac{d}{dt}V(x) \leq 0$ with equality if and only if $x = 0$ (negative definite).

Then *V(x)* is called a Lyapunov function candidate and the system is asymptotically stable in the sense of Lyapunov (i.s.L.). (Note that $V(0) = 0$ is required; otherwise $V(x) = 1 / (1 + | x |)$ would "prove" that $\dot{x}(t) = x$ is locally stable. An additional condition called "properness" or "radial unboundedness" is required in order to conclude global asymptotic stability.)

It is easier to visualize this method of analysis by thinking of a physical system (e.g. vibrating spring and mass) and considering the energy of such a system. If the system loses energy over time and the energy is never restored then eventually the system must grind to a stop and reach some final resting state. This final state is called the attractor. However, finding a function that gives the precise energy of a physical system can be difficult, and for abstract mathematical systems, economic systems or biological systems, the concept of energy may not be applicable.

Lyapunov's realization was that stability can be proven without requiring knowledge of the true physical energy, providing a Lyapunov function can be found to satisfy the above constraints.

Stability for linear state space models

A linear state space model

$$\dot{\mathbf{x}} = A\mathbf{x}$$

is asymptotically stable (in fact, exponentially stable) if all real parts of the eigenvalues of A are negative. This condition is equivalent to the following one:

$$A^T M + MA + N = 0$$

has a solution where $N = N^T > 0$ and $M = M^T > 0$ (positive definite matrices). (The relevant Lyapunov function is $V(x) = x^T Mx$.)

Correspondingly, a time-discrete linear state space model

$$\mathbf{x}_{t+1} = A\mathbf{x}_t$$

is asymptotically stable (in fact, exponentially stable) if all the eigenvalues of A have a modulus smaller than one.

This latter condition has been generalized to switched systems: a linear switched discrete time system (ruled by a set of matrices $\{A_1, \ldots, A_m\}$)

$$\mathbf{x}_{t+1} = A_{i_t}\mathbf{x}_t, \quad A_{i_t} \in \{A_1, \ldots, A_m\}$$

is asymptotically stable (in fact, exponentially stable) if the joint spectral radius of the set $\{A_1, \ldots, A_m\}$ is smaller than one.

Stability for systems with inputs

A system with inputs (or controls) has the form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u})$$

where the (generally time-dependent) input u(t) may be viewed as a *control*, *external input*, *stimulus*, *disturbance*, or *forcing function*. The study of such systems is the subject of control theory and applied in control engineering. For systems with inputs, one must quantify the effect of inputs on the stability of the system. The main two approaches to this analysis are BIBO stability (for linear systems) and input-to-state (ISS) stability (for nonlinear systems)

Example

Consider an equation, where compared to the Van der Pol oscillator equation the friction term is changed:

$$\ddot{y} + y - \varepsilon \left(\frac{\dot{y}^3}{3} - \dot{y} \right) = 0.$$

The equilibrium is at :$\ddot{y} = y = 0.$

Here is a good example of an unsuccessful try to find a Lyapunov function that proves stability:

Let

$$x_1 = y, x_2 = \dot{y}$$

so that the corresponding system is

$$\dot{x_2} = -x_1 + \varepsilon \left(\frac{{x_2}^3}{3} - x_2 \right).$$

Let us choose as a Lyapunov function

$$V = \frac{1}{2} \left(x_1^2 + x_2^2 \right)$$

which is clearly positive definite. Its derivative is

$$\begin{aligned} \dot{V} &= x_1 \dot{x}_1 + x_2 \dot{x}_2 \\ &= x_1 x_2 - x_1 x_2 + \varepsilon \left(\frac{x_2^4}{3} - x_2^2 \right) \\ &= -\epsilon \left(x_2^2 - \frac{x_2^4}{3} \right). \end{aligned}$$

It seems that if the parameter $\varepsilon$is positive, stability is asymptotic for $x_2^2 < 3.$ But this is wrong, since $\dot{V}$does not depend on x_1, and will be 0 everywhere on the x_1 axis.

Barbalat's lemma and stability of time-varying systems

Assume that f is function of time only.

- Having $\dot{f}(t) \to 0$ does not imply that *f*(*t*) has a limit at $t \to \infty$
- Having *f*(*t*) approaching a limit as $t \to \infty$ does not imply that $\dot{f}(t) \to 0$.
- Having *f*(*t*) lower bounded and decreasing ($\dot{f} \leq 0$) implies it converges to a limit. But it does not say whether or not $\dot{f} \to 0$ as $t \to \infty$.

Barbalat's Lemma says

> If *f*(*t*) has a finite limit as $t \to \infty$ and if $\dot{f}$ is uniformly continuous (or $\ddot{f}$ is bounded), then $\dot{f}(t) \to 0$ as $t \to \infty$.

Usually, it is difficult to analyze the *asymptotic* stability of time-varying systems because it is very difficult to find Lyapunov functions with a *negative definite* derivative.

We know that in case of autonomous (time-invariant) systems, if $\dot{V}$ is negative semi-definite (NSD), then also, it is possible to know the asymptotic behaviour by invoking invariant-set theorems. However, this flexibility is not available for *time-varying* systems. This is where "Barbalat's lemma" comes into picture. It says:

> IF *V*(*x*,*t*) satisfies following conditions:
>
> 1. *V*(*x*,*t*) is lower bounded
> 2. $\dot{V}(x,t)$ is negative semi-definite (NSD)
> 3. $\dot{V}(x,t)$ is uniformly continuous in time (satisfied if $\ddot{V}$ is finite)
>
> then $\dot{V}(x,t) \to 0$ as $t \to \infty$.

The following example is taken from page 125 of Slotine and Li's book *Applied Nonlinear Control*.

Consider a non-autonomous system

$$\dot{e} = -e + g \cdot w(t)$$
$$\dot{g} = -e \cdot w(t).$$

This is non-autonomous because the input w is a function of time. Assume that the input $w(t)$ is bounded.

Taking $V = e^2 + g^2$ gives $\dot{V} = -2e^2 \leq 0.$

This says that $V(t) <= V(0)$ by first two conditions and hence e and g are bounded. But it does not say anything about the convergence of e to zero. Moreover, the invariant set theorem cannot be applied, because the dynamics is non-autonomous.

Using Barbalat's lemma:

$$\ddot{V} = -4e(-e + g \cdot w).$$

This is bounded because e, g and w are bounded. This implies $\dot{V} \to 0$ as $t \to \infty$ and hence $e \to 0$. This proves that the error converges.

Chapter- 3

Exponential Stability & Marginal Stability

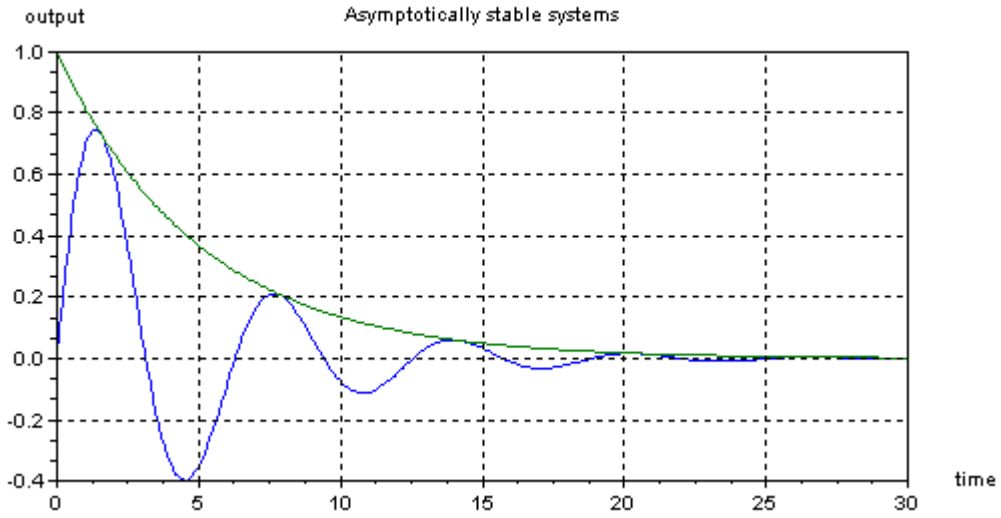
Exponential stability

In control theory, a continuous linear time-invariant system is **exponentially stable** if and only if the system has eigenvalues (i.e., the poles of input-to-output systems) with strictly negative real parts. (i.e., in the left half of the complex plane). A discrete-time input-to-output LTI system is exponentially stable if and only if the poles of its transfer function lie strictly within the unit circle centered on the origin of the complex plane. Exponential stability is a form of asymptotic stability. Systems that are not LTI are exponentially stable if their convergence is bounded by exponential decay.

Practical consequences

An exponentially stable LTI system is one that will not "blow up" (i.e., give an unbounded output) when given a finite input or non-zero initial condition. Moreover, if the system is given a fixed, finite input (i.e., a step), then any resulting oscillations in the output will decay at an exponential rate, and the output will tend asymptotically to a new final, steady-state value. If the system is instead given a Dirac delta impulse as input, then induced oscillations will die away and the system will return to its previous value. If oscillations do not die away, or the system does not return to its original output when an impulse is applied, the system is instead marginally stable.

Example exponentially stable LTI systems



The impulse responses of two exponentially stable systems

The graph on the right shows the impulse response of two similar systems. The green curve is the response of the system with impulse response $y(t) = e^{-\frac{t}{5}}$, while the blue represents the system $y(t) = e^{-\frac{t}{5}} \sin(t)$. Although one response is oscillatory, both return to the original value of 0 over time.

Real-world example

Imagine putting a marble in a ladle. It will settle itself into the lowest point of the ladle and, unless disturbed, will stay there. Now imagine giving the ball a push, which is an approximation to a Dirac delta impulse. The marble will roll back and forth but eventually resettle in the bottom of the ladle. Drawing the horizontal position of the marble over time would give a gradually diminishing sinusoid rather like the blue curve in the image above.

A step input in this case requires supporting the marble away from the bottom of the ladle, so that it cannot roll back. It will stay in the same position and will not, as would be the case if the system were only marginally stable or entirely unstable, continue to move away from the bottom of the ladle under this constant force equal to its weight.

It is important to note that in this example the system is not stable for all inputs. Give the marble a big enough push, and it will fall out of the ladle and fall, stopping only when it reaches the floor. For some systems, therefore, it is proper to state that a system is exponentially stable *over a certain range of inputs*.

Marginal stability

In the theory of dynamical systems, and control theory, a continuous linear time-invariant system is **marginally stable** if and only if the real part of every eigenvalue (or pole) in the system's transfer-function is non-positive, and all eigenvalues with zero real value are simple roots (i.e. the eigenvalues on the imaginary axis are all distinct from one another). If all the poles have strictly negative real parts, the system is instead asymptotically stable.

A discrete linear time-invariant system is marginally stable if and only if the transfer function's spectral radius is 1. That is, the greatest magnitude of any of the eigenvalues (or poles) of the transfer function is 1. The values of the poles must also be distinct. If the spectral radius is less than 1, the system is instead asymptotically stable.

Practical Consequences

A marginally stable system is one that, if given an impulse of finite magnitude as input, will not "blow up" and give an unbounded output. However, oscillations in the output will persist indefinitely, and so there will, in general, be no final steady-state output. If the system is given a step as an input, the system's output will increase indefinitely, with the system effectively acting as an integrator on the input, and so a marginally stable system is not a Bounded Input/Bounded Output system (the information in this para must be verified from other sources).

A system having imaginary poles, i.e having zero real part in the pole(s), will produce sustained oscillations in the output. For example a undamped second order system such as suspension system of your car (mass-spring-damper), from where damper has been removed and spring is ideal i.e. no friction is there, then in theory your car will oscillate forever when you will hit a bump. A system with pole at the origin is also marginally stable but in this case there will be no oscillation in the response as the imaginary part is also zero(jw = 0 means w = 0 rad/sec).

Chapter- 4

Negative Feedback Amplifier

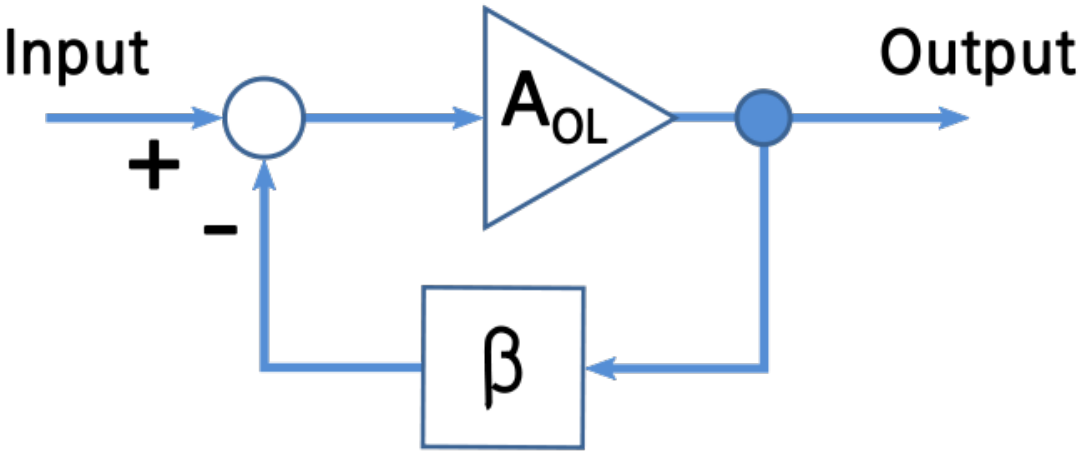


Figure 1: Ideal negative feedback model

A **negative feedback amplifier** (or more commonly simply a **feedback amplifier**) is an amplifier a fraction of the output of which is combined with the input so that a negative feedback opposes the original signal. The applied negative feedback improves performance (gain stability, linearity, frequency response, step response) and reduces sensitivity to parameter variations due to manufacturing or environment. Because of these advantages, negative feedback is used in this way in many amplifiers and control systems.

A negative feedback amplifier is a system of three elements (see Figure 1): an *amplifier* with gain A_{OL}, an attenuating *feedback network* with a constant $\beta < 1$ and a summing circuit acting as a *subtractor* (the circle in the figure). The amplifier is the only obligatory; the other elements may be omitted in some cases. For example, in a voltage (emitter, source, op-amp) follower the feedback network and the summing circuit are not necessary.

Overview

Fundamentally, all electronic devices used to provide power gain (e.g. vacuum tubes, bipolar transistors, MOS transistors) are nonlinear. Negative feedback allows gain to be traded for higher linearity (reducing distortion), amongst other things. If not designed correctly amplifiers with negative feedback can become unstable, resulting in unwanted behavior, such as oscillation. The Nyquist stability criterion developed by Harry Nyquist of Bell Laboratories can be used to study the stability of feedback amplifiers.

Feedback amplifiers share these properties:

Pros:

- Can increase or decrease input impedance (depending on type of feedback)
- Can increase or decrease output impedance (depending on type of feedback)
- Reduces distortion (increases linearity)
- Increases the bandwidth
- Desensitizes gain to component variations
- Can control step response of amplifier

Cons:

- May lead to instability if not designed carefully
- The gain of the amplifier decreases
- The input and output impedances of the amplifier with feedback (the **closed-loop amplifier**) become sensitive to the gain of the amplifier without feedback (the **open-loop amplifier**); that exposes these impedances to variations in the open loop gain, for example, due to parameter variations or due to nonlinearity of the open-loop gain

History

The negative feedback amplifier was invented by Harold Stephen Black (US patent 2,102,671 (issued in 1937)) while a passenger on the Lackawanna Ferry (from Hoboken Terminal to Manhattan) on his way to work at Bell Laboratories (historically located in Manhattan instead of New Jersey in 1927) on August 2, 1927. Black had been toiling at reducing distortion in repeater amplifiers used for telephone transmission. On a blank space in his copy of The New York Times, he recorded the diagram found in Figure 1, and the equations derived below. Black submitted his invention to the U. S. Patent Office on August 8, 1928, and it took more than nine years for the patent to be issued. Black later wrote: "One reason for the delay was that the concept was so contrary to established beliefs that the Patent Office initially did not believe it would work."

Classical feedback

Gain reduction

Below, the voltage gain of the amplifier with feedback, the **closed-loop gain** A_{fb}, is derived in terms of the gain of the amplifier without feedback, the **open-loop gain** A_{OL} and the **feedback factor** β, which governs how much of the output signal is applied to the input. See Figure 1, top right. The open-loop gain A_{OL} in general may be a function of both frequency and voltage; the feedback parameter β is determined by the feedback network that is connected around the amplifier. For an operational amplifier two resistors forming a voltage divider may be used for the feedback network to set β between 0 and 1. This network may be modified using reactive elements like capacitors or inductors to (a) give frequency-dependent closed-loop gain as in equalization/tone-control circuits or (b) construct oscillators. The gain of the amplifier with feedback is derived below in the case of a voltage amplifier with voltage feedback.

Without feedback, the input voltage V'_{in} is applied directly to the amplifier input. The according output voltage is

$$V_{out} = A_{OL} \cdot V'_{in}$$

Suppose now that an attenuating feedback loop applies a fraction $\beta.V_{out}$ of the output to one of the subtractor inputs so that it subtracts from the circuit input voltage V_{in} applied to the other subtractor input. The result of subtraction applied to the amplifier input is

$$V'_{in} = V_{in} - \beta \cdot V_{out}$$

Substituting for V'_{in} in the first expression,

$$V_{out} = A_{OL}(V_{in} - \beta \cdot V_{out})$$

Rearranging

$$V_{out}(1 + \beta \cdot A_{OL}) = V_{in} \cdot A_{OL}$$

Then the gain of the amplifier with feedback, called the closed-loop gain, A_{fb} is given by,

$$A_{\text{fb}} = \frac{V_{\text{out}}}{V_{\text{in}}} = \frac{A_{OL}}{1 + \beta \cdot A_{OL}}$$

If $A_{\text{OL}} >> 1$, then $A_{\text{fb}} \approx 1 / \beta$ and the effective amplification (or closed-loop gain) A_{fb} is set by the feedback constant β, and hence set by the feedback network, usually a simple reproducible network, thus making linearizing and stabilizing the amplification characteristics straightforward. Note also that if there are conditions where $\beta A_{\text{OL}} = -1$, the amplifier has infinite amplification – it has become an oscillator, and the system is

unstable. The stability characteristics of the gain feedback product βA_{OL} are often displayed and investigated on a Nyquist plot (a polar plot of the gain/phase shift as a parametric function of frequency). A simpler, but less general technique, uses Bode plots.

The combination $L = \beta A_{OL}$ appears commonly in feedback analysis and is called the **loop gain**. The combination $(1 + \beta A_{OL})$ also appears commonly and is variously named as the **desensitivity factor** or the **improvement factor**.

Bandwidth extension

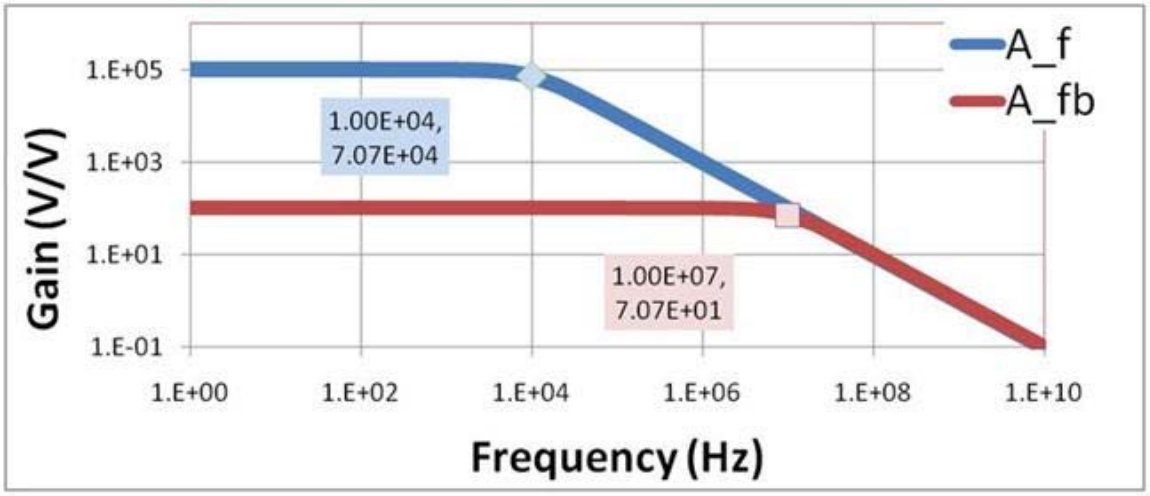


Figure 2: Gain vs. frequency for a single-pole amplifier with and without feedback; corner frequencies are labeled.

Feedback can be used to extend the bandwidth of an amplifier (speed it up) at the cost of lowering the amplifier gain. Figure 2 shows such a comparison. The figure is understood as follows. Without feedback the so-called **open-loop** gain in this example has a single time constant frequency response given by

$$A_{OL}(f) = \frac{A_0}{1 + jf/f_C} \ ,$$

where f_C is the cutoff or corner frequency of the amplifier: in this example $f_C = 10^4$ Hz and the gain at zero frequency $A_0 = 10^5$ V/V. The figure shows the gain is flat out to the corner frequency and then drops. When feedback is present the so-called **closed-loop** gain, as shown in the formula of the previous section, becomes,

$$A_{fb}(f) = \frac{A_{OL}}{1 + \beta A_{OL}} = \frac{A_0/(1 + jf/f_C)}{1 + \beta A_0/(1 + jf/f_C)}$$

$$= \frac{A_0}{1 + jf/f_C + \beta A_0}$$
$$= \frac{A_0}{(1 + \beta A_0)\left(1 + j\frac{f}{(1+\beta A_0) f_C}\right)} .$$

The last expression shows the feedback amplifier still has a single time constant behavior, but the corner frequency is now increased by the improvement factor (1 + β A_0), and the gain at zero frequency has dropped by exactly the same factor. This behavior is called the **gain-bandwidth tradeoff**. In Figure 2, (1 + β A_0) = 10^3, so $A_{fb}(0)$= 10^5 / 10^3 = 100 V/V, and f_C increases to $10^4 \times 10^3 = 10^7$ Hz.

Multiple poles

When the open-loop gain has several poles, rather than the single pole of the above example, feedback can result in complex poles (real and imaginary parts). In a two-pole case, the result is peaking in the frequency response of the feedback amplifier near its corner frequency, and ringing and overshoot in its step response. In the case of more than two poles, the feedback amplifier can become unstable, and oscillate.

Asymptotic gain model

In the above analysis the feedback network is unilateral. However, real feedback networks often exhibit **feed forward** as well, that is, they feed a small portion of the input to the output, degrading performance of the feedback amplifier. A more general way to model negative feedback amplifiers including this effect is with the asymptotic gain model.

Feedback and amplifier type

Amplifiers use current or voltage as input and output, so four types of amplifier are possible. Any of these four choices may be the open-loop amplifier used to construct the feedback amplifier. The objective for the feedback amplifier also may be any one of the four types of amplifier, not necessarily the same type as the open-loop amplifier. For example, an op amp (voltage amplifier) can be arranged to make a current amplifier instead. The conversion from one type to another is implemented using different feedback connections, usually referred to as series or shunt (parallel) connections. See the table below.

Feedback amplifier type	Input connection	Output connection	Ideal feedback	Two-port feedback
Current	**Shunt**	**Series**	**CCCS**	**g-parameter**
Transresistance	**Shunt**	**Shunt**	**CCVS**	**y-parameter**
Transconductance	**Series**	**Series**	**VCCS**	**z-parameter**

Voltage **Series** **Shunt** **VCVS** **h-parameter**

The feedback can be implemented using a two-port network. There are four types of two-port network, and the selection depends upon the type of feedback. For example, for a current feedback amplifier, current at the output is sampled and combined with current at the input. Therefore, the feedback ideally is performed using an (output) current-controlled current source (CCCS), and its imperfect realization using a two-port network also must incorporate a CCCS, that is, the appropriate choice for feedback network is a g-parameter two-port.

Two-port analysis of feedback

One approach to feedback is the use of return ratio. Here an alternative method used in most textbooks is presented by means of an example treated in the article on asymptotic gain model.

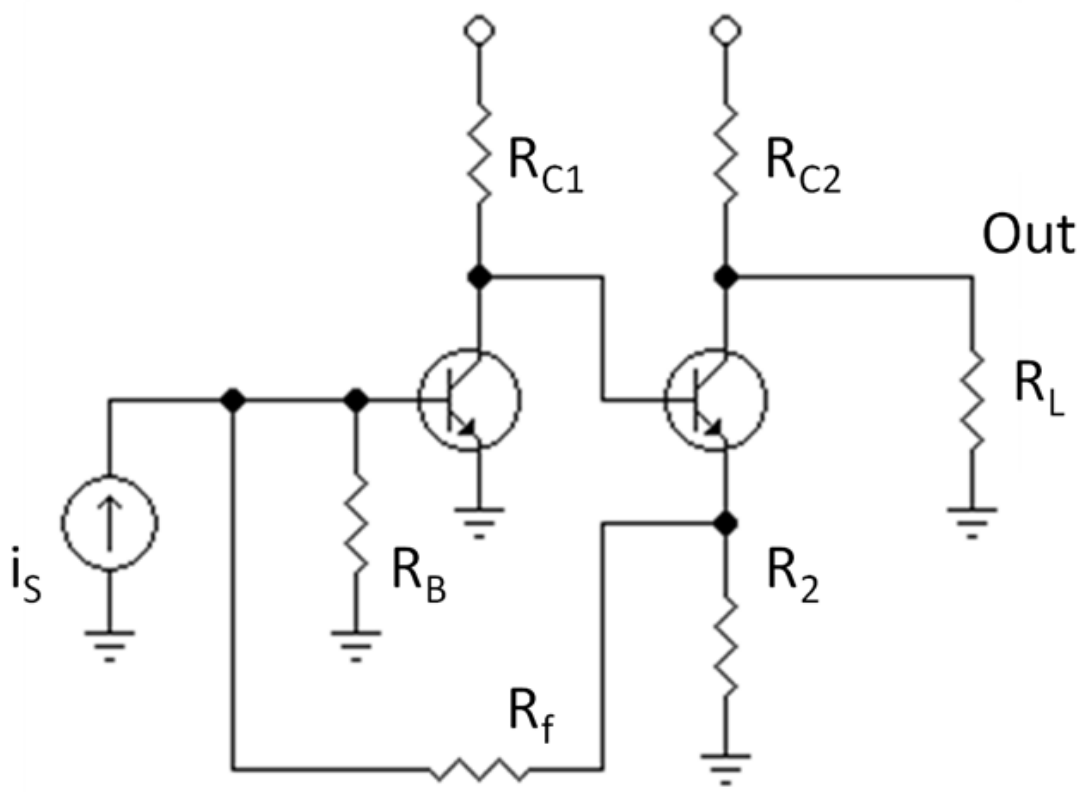


Figure 3: A *shunt-series* feedback amplifier

Figure 3 shows a two-transistor amplifier with a feedback resistor R_f. The aim is to analyze this circuit to find three items: the gain, the output impedance looking into the amplifier from the load, and the input impedance looking into the amplifier from the source.

Replacement of the feedback network with a two-port

The first step is replacement of the feedback network by a two-port. Just what components go into the two-port?

On the input side of the two-port we have R_f. If the voltage at the right side of R_f changes, it changes the current in R_f that is subtracted from the current entering the base of the input transistor. That is, the input side of the two-port is a dependent current source controlled by the voltage at the top of resistor R_2.

One might say the second stage of the amplifier is just a voltage follower, transmitting the voltage at the collector of the input transistor to the top of R_2. That is, the monitored output signal is really the voltage at the collector of the input transistor. That view is legitimate, but then the voltage follower stage becomes part of the feedback network. That makes analysis of feedback more complicated.

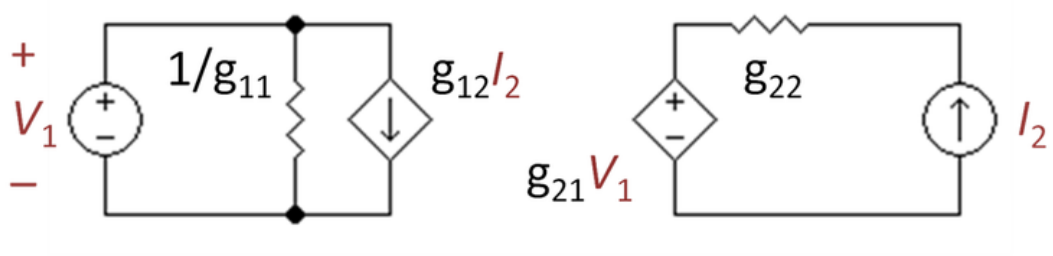


Figure 4: The g-parameter feedback network

An alternative view is that the voltage at the top of R_2 is set by the emitter current of the output transistor. That view leads to an entirely passive feedback network made up of R_2 and R_f. The variable controlling the feedback is the emitter current, so the feedback is a current-controlled current source (CCCS). We search through the four available two-port networks and find the only one with a CCCS is the g-parameter two-port, shown in Figure 4. The next task is to select the g-parameters so that the two-port of Figure 4 is electrically equivalent to the L-section made up of R_2 and R_f. That selection is an algebraic procedure made most simply by looking at two individual cases: the case with $V_1 = 0$, which makes the VCVS on the right side of the two-port a short-circuit; and the case with $I_2 = 0$. which makes the CCCS on the left side an open circuit. The algebra in these two cases is simple, much easier than solving for all variables at once. The choice of g-parameters that make the two-port and the L-section behave the same way are shown in the table below.

g_{11}	g_{12}	g_{21}	g_{22}
$\frac{1}{R_f + R_2}$	$-\frac{R_2}{R_2 + R_f}$	$\frac{R_2}{R_2 + R_f}$	$R_2 // R_f$

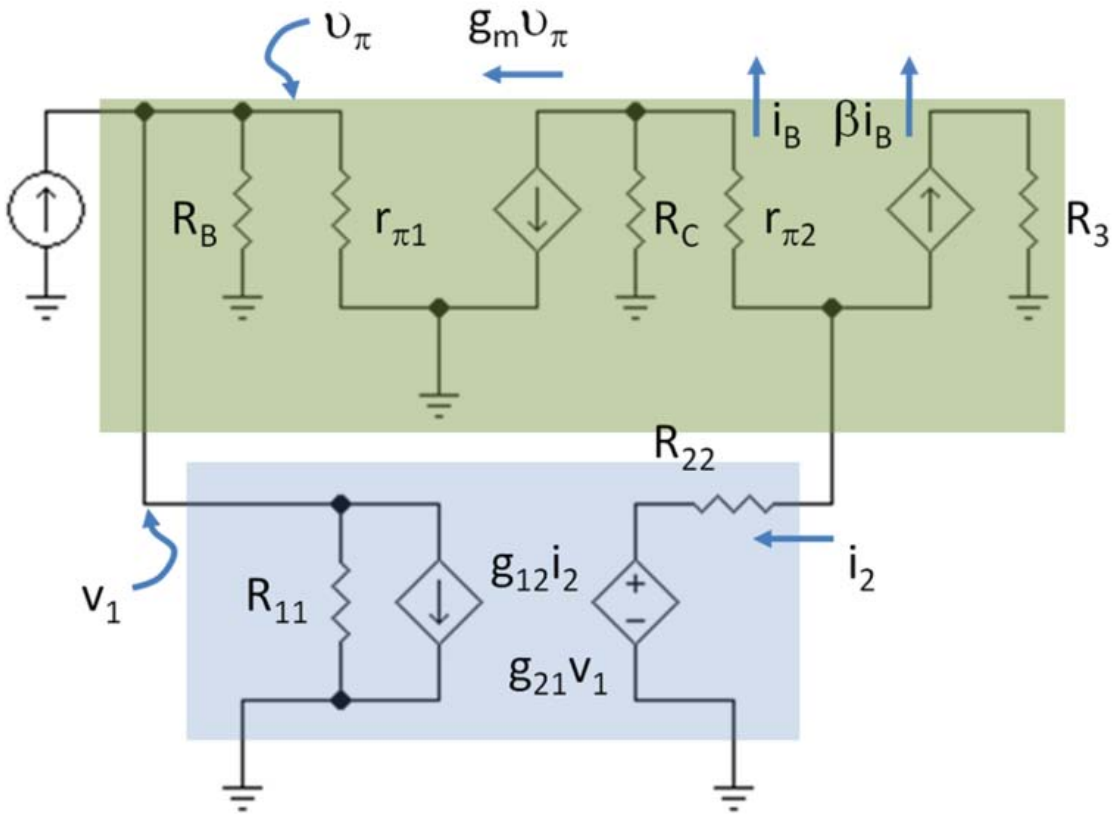


Figure 5: Small-signal circuit with two-port for feedback network; upper shaded box: main amplifier; lower shaded box: feedback two-port replacing the *L*-section made up of R_f and R_2.

Small-signal circuit

The next step is to draw the small-signal schematic for the amplifier with the two-port in place using the hybrid-pi model for the transistors. Figure 5 shows the schematic with notation $R_3 = R_{C2} \,//\, R_L$ and $R_{11} = 1 / g_{11}$, $R_{22} = g_{22}$.

Loaded open-loop gain

Figure 3 indicates the output node, but not the choice of output variable. A useful choice is the short-circuit current output of the amplifier (leading to the short-circuit current gain). Because this variable leads simply to any of the other choices (for example, load voltage or load current), the short-circuit current gain is found below.

First the loaded **open-loop gain** is found. The feedback is turned off by setting $g_{12} = g_{21} = 0$. The idea is to find how much the amplifier gain is changed because of the resistors in the feedback network by themselves, with the feedback turned off. This calculation is pretty easy because R_{11}, R_B, *and* $r_{\pi 1}$ all are in parallel and $v_1 = v_\pi$. Let $R_1 = R_{11} \,//\, R_B \,//\, r_{\pi 1}$. In addition, $i_2 = -(\beta+1)\, i_B$. The result for the open-loop current gain A_{OL} is:

$$A_{OL} = \frac{\beta i_B}{i_S} = g_m R_C \left(\frac{\beta}{\beta+1}\right)\left(\frac{R_1}{R_{22} + \frac{r_{\pi 2}+R_C}{\beta+1}}\right) .$$

Gain with feedback

In the classical approach to feedback, the feedforward represented by the VCVS (that is, $g_{21}\, v_1$) is neglected. That makes the circuit of Figure 5 resemble the block diagram of Figure 1, and the gain with feedback is then:

$$A_{FB} = \frac{A_{OL}}{1+\beta_{FB}A_{OL}} = \frac{A_{OL}}{1+\frac{R_2}{R_2+R_f}A_{OL}} ,$$

where the feedback factor $\beta_{FB} = -g_{12}$. Notation β_{FB} is introduced for the feedback factor to distinguish it from the transistor β.

Input and output resistances

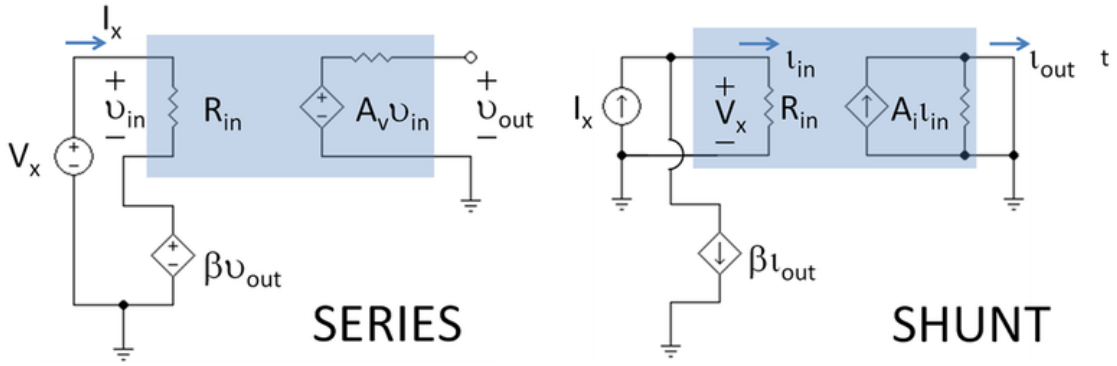


Figure 6: Circuit set-up for finding feedback amplifier input resistance

First, a digression on how two-port theory approaches resistance determination, and then its application to the amplifier at hand.

Background on resistance determination

Figure 6 shows an equivalent circuit for finding the input resistance of a feedback voltage amplifier (left) and for a feedback current amplifier (right). These arrangements are typical Miller theorem applications.

In the case of the voltage amplifier, the output voltage βV_{out} of the feedback network is applied in series and with an opposite polarity to the input voltage V_x travelling over the loop (but in respect to ground, the polarities are the same). As a result, the effective

voltage across and the current through the amplifier input resistance R_{in} decrease so that the circuit input resistance increases (one might say that R_{in} apparently increases). Its new value can be calculated by applying Miller theorem (for voltages) or the basic circuit laws. Thus Kirchhoff's voltage law provides:

$$V_x = I_x R_{in} + \beta v_{out} \ ,$$

where $v_{out} = A_v \, v_{in} = A_v \, I_x \, R_{in}$. Substituting this result in the above equation and solving for the input resistance of the feedback amplifier, the result is:

$$R_{in}(fb) = \frac{V_x}{I_x} = (1 + \beta A_v) \, R_{in} \ .$$

The general conclusion to be drawn from this example and a similar example for the output resistance case is:

A series feedback connection at the input (output) increases the input (output) resistance by a factor (1 + β A_{OL}), where A_{OL} = open loop gain.

On the other hand, for the current amplifier, the output current βI_{out} of the feedback network is applied in parallel and with an opposite direction to the input current I_x. As a result, the total current flowing through the circuit input (not only through the input resistance R_{in}) increases and the voltage across it decreases so that the circuit input resistance decreases (R_{in} apparently decreases). Its new value can be calculated by applying the dual Miller theorem (for currents) or the basic Kirchhoff's laws:

$$I_x = \frac{V_{in}}{R_{in}} + \beta i_{out} \ .$$

where $i_{out} = A_i \, i_{in} = A_i \, V_x \, / \, R_{in}$. Substituting this result in the above equation and solving for the input resistance of the feedback amplifier, the result is:

$$R_{in}(fb) = \frac{V_x}{I_x} = \frac{R_{in}}{(1 + \beta A_i)} \ .$$

The general conclusion to be drawn from this example and a similar example for the output resistance case is:

A parallel feedback connection at the input (output) decreases the input (output) resistance by a factor (1 + β A_{OL}), where A_{OL} = open loop gain.

These conclusions can be generalized to treat cases with arbitrary Norton or Thévenin drives, arbitrary loads, and general two-port feedback networks. However, the results do depend upon the main amplifier having a representation as a two-port – that is, the results

depend on the *same* current entering and leaving the input terminals, and likewise, the same current that leaves one output terminal must enter the other output terminal.

A broader conclusion to be drawn, independent of the quantitative details, is that feedback can be used to increase or to decrease the input and output impedances.

Application to the example amplifier

These resistance results now are applied to the amplifier of Figure 3 and Figure 5. The *improvement factor* that reduces the gain, namely ($1 + \beta_{FB} A_{OL}$), directly decides the effect of feedback upon the input and output resistances of the amplifier. In the case of a shunt connection, the input impedance is reduced by this factor; and in the case of series connection, the impedance is multiplied by this factor. However, the impedance that is modified by feedback is the impedance of the amplifier in Figure 5 with the feedback turned off, and does include the modifications to impedance caused by the resistors of the feedback network.

Therefore, the input impedance seen by the source with feedback turned off is $R_{in} = R_1 = R_{11} // R_B // r_{\pi 1}$, and with the feedback turned on (but no feedforward)

$$R_{in} = \frac{R_1}{1 + \beta_{FB} A_{OL}} ,$$

where *division* is used because the input connection is *shunt*: the feedback two-port is in parallel with the signal source at the input side of the amplifier. A reminder: A_{OL} is the *loaded* open loop gain found above, as modified by the resistors of the feedback network.

The impedance seen by the load needs further discussion. The load in Figure 5 is connected to the collector of the output transistor, and therefore is separated from the body of the amplifier by the infinite impedance of the output current source. Therefore, feedback has no effect on the output impedance, which remains simply R_{C2} as seen by the load resistor R_L in Figure 3.

If instead we wanted to find the impedance presented at the *emitter* of the output transistor (instead of its collector), which is series connected to the feedback network, feedback would increase this resistance by the improvement factor ($1 + \beta_{FB} A_{OL}$).

Load voltage and load current

The gain derived above is the current gain at the collector of the output transistor. To relate this gain to the gain when voltage is the output of the amplifier, notice that the output voltage at the load R_L is related to the collector current by Ohm's law as $v_L = i_C (R_{C2} // R_L)$. Consequently, the transresistance gain v_L / i_S is found by multiplying the current gain by $R_{C2} // R_L$:

$$\frac{v_L}{i_S} = A_{FB}(R_{C2}//R_L) \ .$$

Similarly, if the output of the amplifier is taken to be the current in the load resistor R_L, current division determines the load current, and the gain is then:

$$\frac{i_L}{i_S} = A_{FB}\frac{R_{C2}}{R_{C2}+R_L} \ .$$

Is the main amplifier block a two port?

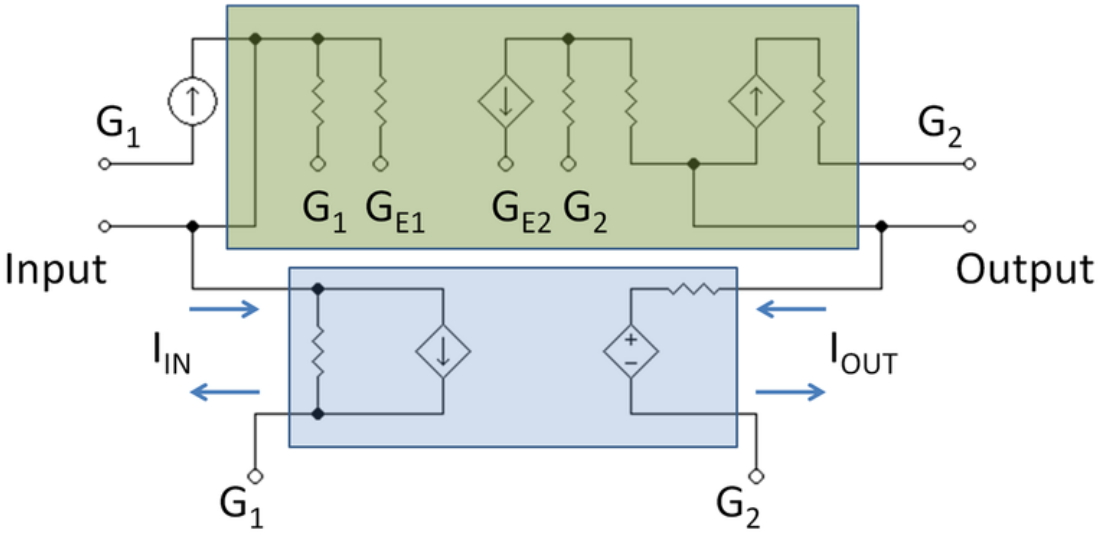


Figure 7: Amplifier with ground connections labeled by G. The feedback network satisfies the port conditions.

Some complications follow, intended for the attentive reader.

Figure 7 shows the small-signal schematic with the main amplifier and the feedback two-port in shaded boxes. The two-port satisfies the port conditions: at the input port, I_{in} enters and leaves the port, and likewise at the output, I_{out} enters and leaves. The main amplifier is shown in the upper shaded box. The ground connections are labeled.

Figure 7 shows the interesting fact that the main amplifier does not satisfy the port conditions at its input and output unless the ground connections are chosen to make that happen. For example, on the input side, the current entering the main amplifier is I_S. This current is divided three ways: to the feedback network, to the bias resistor R_B and to the base resistance of the input transistor r_π. To satisfy the port condition for the main amplifier, all three components must be returned to the input side of the main amplifier, which means all the ground leads labeled G_1 must be connected, as well as emitter lead G_{E1}. Likewise, on the output side, all ground connections G_2 must be connected and also ground connection G_{E2}. Then, at the bottom of the schematic, underneath the feedback two-port and outside the amplifier blocks, G_1 is connected to G_2. That forces the ground

currents to divide between the input and output sides as planned. Notice that this connection arrangement *splits the emitter* of the input transistor into a base-side and a collector-side – a physically impossible thing to do, but electrically the circuit sees all the ground connections as one node, so this fiction is permitted.

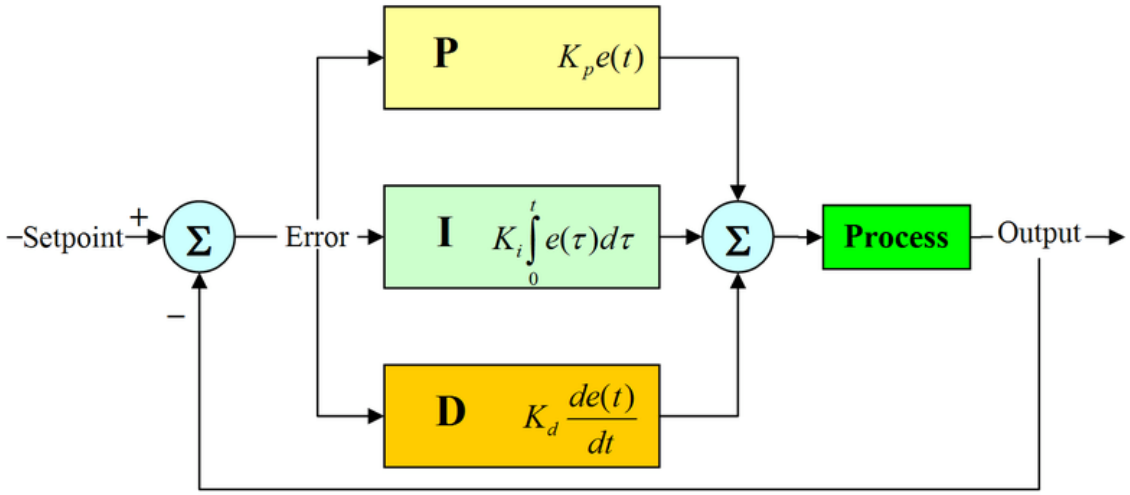
Of course, the way the ground leads are connected makes no difference to the amplifier (they are all one node), but it makes a difference to the port conditions. That is a weakness of this approach: the port conditions are needed to justify the method, but the circuit really is unaffected by how currents are traded among ground connections.

However, if there is **no possible arrangement** of ground conditions that will lead to the port conditions, the circuit might not behave the same way. The improvement factors ($1 + \beta_{FB} A_{OL}$) for determining input and output impedance might not work. This situation is awkward, because a failure to make a two-port may reflect a real problem (it just is not possible), or reflect a lack of imagination (for example, just did not think of splitting the emitter node in two). As a consequence, when the port conditions are in doubt, at least two approaches are possible to establish whether improvement factors are accurate: either simulate an example using Spice and compare results with use of an improvement factor, or calculate the impedance using a test source and compare results.

A more radical choice is to drop the two-port approach altogether, and use return ratios. That choice might be advisable if small-signal device models are complex, or are not available (for example, the devices are known only numerically, perhaps from measurement or from SPICE simulations).

Chapter- 5

PID Controller



A block diagram of a PID controller

A **proportional–integral–derivative controller** (**PID controller**) is a generic control loop feedback mechanism (controller) widely used in industrial control systems – a PID is the most commonly used feedback controller. A PID controller calculates an "error" value as the difference between a measured process variable and a desired setpoint. The controller attempts to minimize the error by adjusting the process control inputs.

The PID controller calculation (algorithm) involves three separate parameters, and is accordingly sometimes called **three-term control**: the proportional, the integral and derivative values, denoted *P, I,* and *D.* Heuristically, these values can be interpreted in terms of time: *P* depends on the *present* error, *I* on the accumulation of *past* errors, and *D* is a prediction of *future* errors, based on current rate of change. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve or the power supply of a heating element.

In the absence of knowledge of the underlying process, a PID controller is the best controller. By tuning the three constants in the PID controller algorithm, the controller

can provide control action designed for specific process requirements. The response of the controller can be described in terms of the responsiveness of the controller to an error, the degree to which the controller overshoots the setpoint and the degree of system oscillation. Note that the use of the PID algorithm for control does not guarantee optimal control of the system or system stability.

Some applications may require using only one or two modes to provide the appropriate system control. This is achieved by setting the gain of undesired control outputs to zero. A PID controller will be called a PI, PD, P or I controller in the absence of the respective control actions. PI controllers are fairly common, since derivative action is sensitive to measurement noise, whereas the absence of an integral value may prevent the system from reaching its target value due to the control action.

Control loop basics

A familiar example of a control loop is the action taken when adjusting hot and cold faucet valves to maintain the faucet water at the desired temperature. This typically involves the mixing of two process streams, the hot and cold water. The person touches the water to sense or measure its temperature. Based on this feedback they perform a control action to adjust the hot and cold water valves until the process temperature stabilizes at the desired value.

Sensing water temperature is analogous to taking a measurement of the process value or process variable (PV). The desired temperature is called the setpoint (SP). The input to the process (the water valve position) is called the manipulated variable (MV). The difference between the temperature measurement and the setpoint is the error (e) and quantifies whether the water is too hot or too cold and by how much.

After measuring the temperature (PV), and then calculating the error, the controller decides when to change the tap position (MV) and by how much. When the controller first turns the valve on, it may turn the hot valve only slightly if warm water is desired, or it may open the valve all the way if very hot water is desired. This is an example of a simple **proportional** control. In the event that hot water does not arrive quickly, the controller may try to speed-up the process by opening up the hot water valve more-and-more as time goes by. This is an example of an **integral** control.

Making a change that is too large when the error is small is equivalent to a high gain controller and will lead to overshoot. If the controller were to repeatedly make changes that were too large and repeatedly overshoot the target, the output would oscillate around the setpoint in either a constant, growing, or decaying sinusoid. If the oscillations increase with time then the system is unstable, whereas if they decrease the system is stable. If the oscillations remain at a constant magnitude the system is marginally stable.

In the interest of achieving a gradual convergence at the desired temperature (SP), the controller may wish to damp the anticipated future oscillations. So in order to compensate

for this effect, the controller may elect to temper their adjustments. This can be thought of as a **derivative** control method.

If a controller starts from a stable state at zero error (PV = SP), then further changes by the controller will be in response to changes in other measured or unmeasured inputs to the process that impact on the process, and hence on the PV. Variables that impact on the process other than the MV are known as disturbances. Generally controllers are used to reject disturbances and/or implement setpoint changes. Changes in feedwater temperature constitute a disturbance to the faucet temperature control process.

In theory, a controller can be used to control any process which has a measurable output (PV), a known ideal value for that output (SP) and an input to the process (MV) that will affect the relevant PV. Controllers are used in industry to regulate temperature, pressure, flow rate, chemical composition, speed and practically every other variable for which a measurement exists.

PID controller theory

This section describes the parallel or non-interacting form of the PID controller.

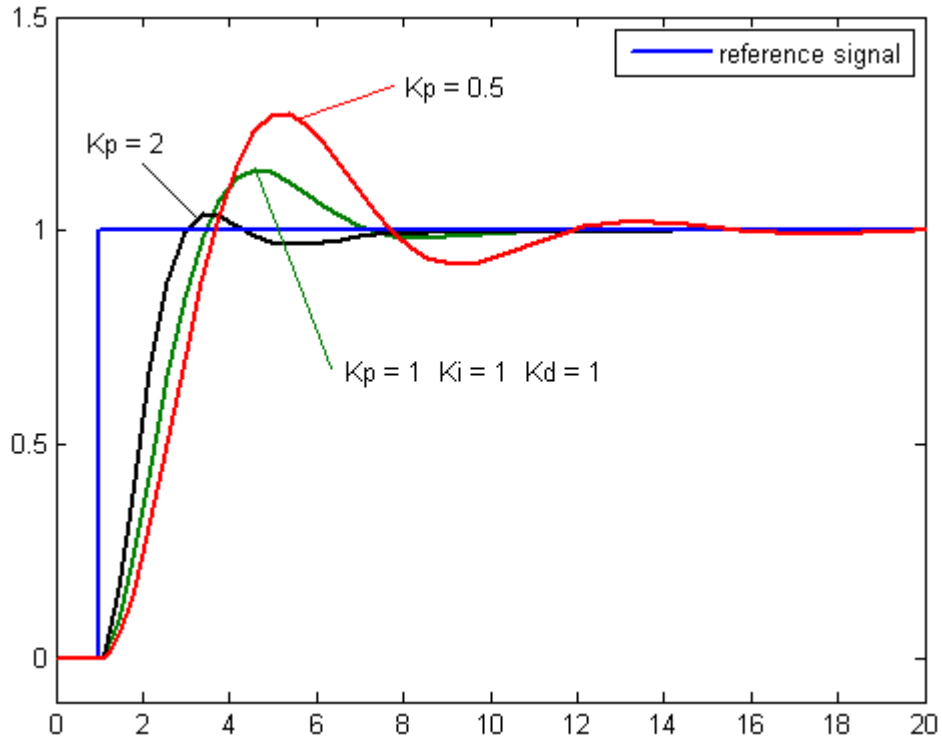
The PID control scheme is named after its three correcting terms, whose sum constitutes the manipulated variable (MV). Hence:

$$\mathrm{MV(t)} = P_{\mathrm{out}} + I_{\mathrm{out}} + D_{\mathrm{out}}$$

where

> P_{out}, I_{out}, and D_{out} are the contributions to the output from the PID controller from each of the three terms, as defined below.

Proportional term



Plot of PV vs time, for three values of K_p (K_i and K_d held constant)

The proportional term (sometimes called *gain*) makes a change to the output that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by a constant K_p, called the proportional gain.

The proportional term is given by:

$$P_{\text{out}} = K_p\, e(t)$$

where

P_{out}: Proportional term of output
K_p: Proportional gain, a tuning parameter
SP: Setpoint, the desired value
PV: Process value (or process variable), the measured value
e: Error = $SP - PV$
t: Time or instantaneous time (the present)

A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can become unstable. In contrast, a small gain results in a small output response to a large input error, and a less responsive

(or sensitive) controller. If the proportional gain is too low, the control action may be too small when responding to system disturbances.

Droop

A pure proportional controller will not always settle at its target value, but may retain a steady-state error. Specifically, the process gain - drift in the absence of control, such as cooling of a furnace towards room temperature, biases a pure proportional controller. If the process gain is down, as in cooling, then the bias will be *below* the set point, hence the term "droop".

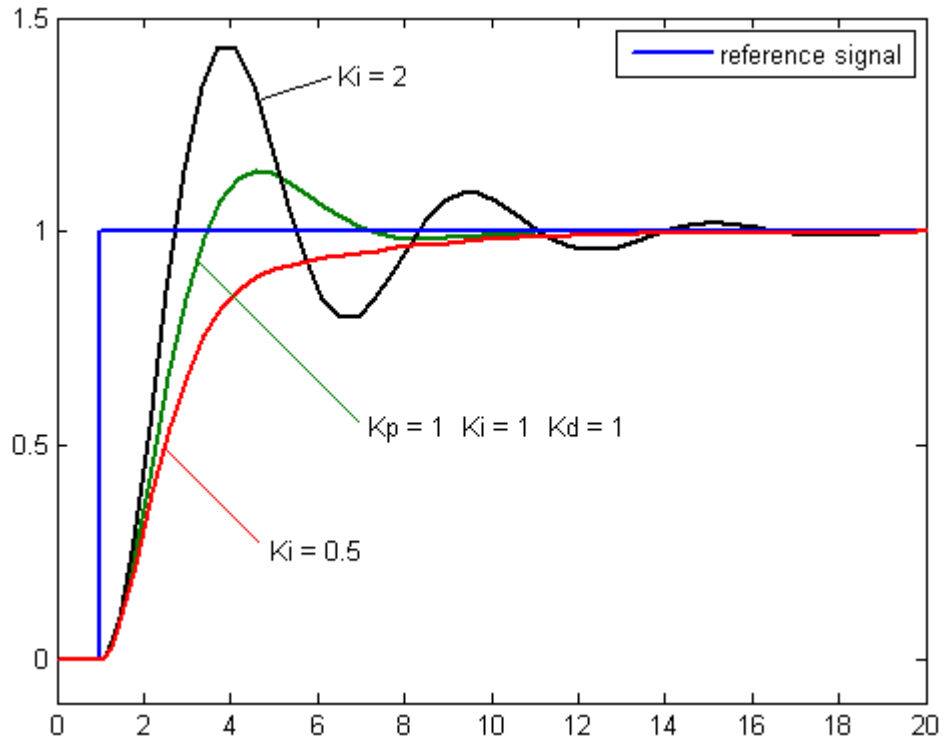
Droop is proportional to process gain and inversely proportional to proportional gain. Specifically the steady-state error is given by:

$$e = G / K_p$$

Droop is an inherent defect of purely proportional control. Droop may be mitigated by adding a compensating *bias* term (setting the setpoint above the true desired value), or corrected by adding an integration term (in a PI or PID controller), which effectively computes a bias adaptively.

Despite droop, both tuning theory and industrial practice indicate that it is the proportional term that should contribute the bulk of the output change.

Integral term



Plot of PV vs time, for three values of K_i (K_p and K_d held constant)

The contribution from the integral term (sometimes called *reset*) is proportional to both the magnitude of the error and the duration of the error. Summing the instantaneous error over time (integrating the error) gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain and added to the controller output. The magnitude of the contribution of the integral term to the overall control action is determined by the integral gain, K_i.

The integral term is given by:

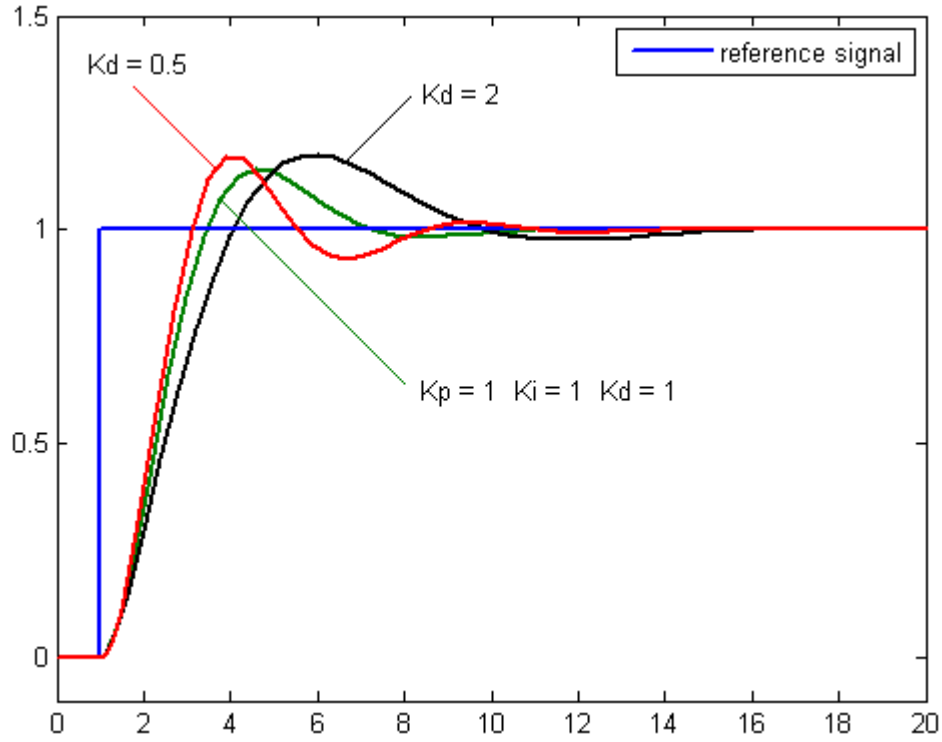
$$I_{\text{out}} = K_i \int_0^t e(\tau)\, d\tau$$

where

I_{out}: Integral term of output
K_i: Integral gain, a tuning parameter
SP: Setpoint, the desired value
PV: Process value (or process variable), the measured value
e: Error = $SP - PV$
t: Time or instantaneous time (the present)
τ: a dummy integration variable

The integral term (when added to the proportional term) accelerates the movement of the process towards setpoint and eliminates the residual steady-state error that occurs with a proportional only controller. However, since the integral term is responding to accumulated errors from the past, it can cause the present value to overshoot the setpoint value (cross over the setpoint and then create a deviation in the other direction).

Derivative term



Plot of PV vs time, for three values of K_d (K_p and K_i held constant)

The rate of change of the process error is calculated by determining the slope of the error over time (i.e., its first derivative with respect to time) and multiplying this rate of change by the derivative gain K_d. The magnitude of the contribution of the derivative term (sometimes called *rate*) to the overall control action is termed the derivative gain, K_d.

The derivative term is given by:

$$D_{\text{out}} = K_d \frac{d}{dt} e(t)$$

where

D_{out}: Derivative term of output
K_d: Derivative gain, a tuning parameter
SP: Setpoint, the desired value
PV: Process value (or process variable), the measured value
e: Error = $SP - PV$
t: Time or instantaneous time (the present)

The derivative term slows the rate of change of the controller output and this effect is most noticeable close to the controller setpoint. Hence, derivative control is used to reduce the magnitude of the overshoot produced by the integral component and improve

the combined controller-process stability. However, differentiation of a signal amplifies noise and thus this term in the controller is highly sensitive to noise in the error term, and can cause a process to become unstable if the noise and the derivative gain are sufficiently large. Hence an approximation to a differentiator with a limited bandwidth is more commonly used. Such a circuit is known as a Phase-Lead compensator.

Summary

The proportional, integral, and derivative terms are summed to calculate the output of the PID controller. Defining $u(t)$ as the controller output, the final form of the PID algorithm is:

$$\mathrm{u(t)} = \mathrm{MV(t)} = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{d}{dt} e(t)$$

where the tuning parameters are:

Proportional gain, K_p
: Larger values typically mean faster response since the larger the error, the larger the proportional term compensation. An excessively large proportional gain will lead to process instability and oscillation.

Integral gain, K_i
: Larger values imply steady state errors are eliminated more quickly. The trade-off is larger overshoot: any negative error integrated during transient response must be integrated away by positive error before reaching steady state.

Derivative gain, K_d
: Larger values decrease overshoot, but slow down transient response and may lead to instability due to signal noise amplification in the differentiation of the error

Loop tuning

Tuning a control loop is the adjustment of its control parameters (gain/proportional band, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response. Stability (bounded oscillation) is a basic requirement, but beyond that, different systems have different behavior, different applications have different requirements, and requirements may conflict with one another.

Some processes have a degree of non-linearity and so parameters that work well at full-load conditions don't work when the process is starting up from no-load; this can be corrected by gain scheduling (using different parameters in different operating regions). PID controllers often provide acceptable control using default tunings, but performance can generally be improved by careful tuning, and performance may be unacceptable with poor tuning.

PID tuning is a difficult problem, even though there are only three parameters and in principle is simple to describe, because it must satisfy complex criteria within the limitations of PID control. There are accordingly various methods for loop tuning, and more sophisticated techniques are the subject of patents; this section describes some traditional manual methods for loop tuning.

Stability

If the PID controller parameters (the gains of the proportional, integral and derivative terms) are chosen incorrectly, the controlled process input can be unstable, i.e. its output diverges, with or without oscillation, and is limited only by saturation or mechanical breakage. Instability is caused by *excess* gain, particularly in the presence of significant lag.

Generally, stability of response (the reverse of instability) is required and the process must not oscillate for any combination of process conditions and setpoints, though sometimes marginal stability (bounded oscillation) is acceptable or desired.

Optimum behavior

The optimum behavior on a process change or setpoint change varies depending on the application.

Two basic requirements are *regulation* (disturbance rejection – staying at a given setpoint) and *command tracking* (implementing setpoint changes) – these refer to how well the controlled variable tracks the desired value. Specific criteria for command tracking include rise time and settling time. Some processes must not allow an overshoot of the process variable beyond the setpoint if, for example, this would be unsafe. Other processes must minimize the energy expended in reaching a new setpoint.

Overview of methods

There are several methods for tuning a PID loop. The most effective methods generally involve the development of some form of process model, then choosing P, I, and D based on the dynamic model parameters. Manual tuning methods can be relatively inefficient, particularly if the loops have response times on the order of minutes or longer.

The choice of method will depend largely on whether or not the loop can be taken "offline" for tuning, and the response time of the system. If the system can be taken offline, the best tuning method often involves subjecting the system to a step change in input, measuring the output as a function of time, and using this response to determine the control parameters.

Choosing a Tuning Method

Method	Advantages	Disadvantages
Manual Tuning	No math required. Online method.	Requires experienced personnel.
Ziegler–Nichols	Proven Method. Online method.	Process upset, some trial-and-error, very aggressive tuning.
Software Tools	Consistent tuning. Online or offline method. May include valve and sensor analysis. Allow simulation before downloading. Can support Non-Steady State (NSS) Tuning.	Some cost and training involved.
Cohen-Coon	Good process models.	Some math. Offline method. Only good for first-order processes.

Manual tuning

If the system must remain online, one tuning method is to first set K_i and K_d values to zero. Increase the K_p until the output of the loop oscillates, then the K_p should be set to approximately half of that value for a "quarter amplitude decay" type response. Then increase K_i until any offset is correct in sufficient time for the process. However, too much K_i will cause instability. Finally, increase K_d, if required, until the loop is acceptably quick to reach its reference after a load disturbance. However, too much K_d will cause excessive response and overshoot. A fast PID loop tuning usually overshoots slightly to reach the setpoint more quickly; however, some systems cannot accept overshoot, in which case an *over-damped* closed-loop system is required, which will require a K_p setting significantly less than half that of the K_p setting causing oscillation.

Effects of *increasing* a parameter independently

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Decrease significantly	Degrade
K_d	Minor decrease	Minor decrease	Minor decrease	No effect in theory	Improve if K_d small

Ziegler–Nichols method

Another heuristic tuning method is formally known as the Ziegler–Nichols method, introduced by John G. Ziegler and Nathaniel B. Nichols in the 1940s. As in the method above, the K_i and K_d gains are first set to zero. The P gain is increased until it reaches the

ultimate gain, K_u, at which the output of the loop starts to oscillate. K_u and the oscillation period P_u are used to set the gains as shown:

Ziegler–Nichols method

Control Type	K_p	K_i	K_d
P	$0.50K_u$	-	-
PI	$0.45K_u$	$1.2K_p / P_u$	-
PID	$0.60K_u$	$2K_p / P_u$	$K_pP_u / 8$

These gains apply to the ideal, parallel form of the PID controller. When applied to the standard PID form, the integral and derivative time parameters T_i and T_d are only dependent on the oscillation period P_u.

PID tuning software

Most modern industrial facilities no longer tune loops using the manual calculation methods shown above. Instead, PID tuning and loop optimization software are used to ensure consistent results. These software packages will gather the data, develop process models, and suggest optimal tuning. Some software packages can even develop tuning by gathering data from reference changes.

Mathematical PID loop tuning induces an impulse in the system, and then uses the controlled system's frequency response to design the PID loop values. In loops with response times of several minutes, mathematical loop tuning is recommended, because trial and error can literally take days just to find a stable set of loop values. Optimal values are harder to find. Some digital loop controllers offer a self-tuning feature in which very small setpoint changes are sent to the process, allowing the controller itself to calculate optimal tuning values.

Other formulas are available to tune the loop according to different performance criteria. Many patented formulas are now embedded within PID tuning software and hardware modules.

Advances in automated PID Loop Tuning software also deliver algorithms for tuning PID Loops in a dynamic or Non-Steady State (NSS) scenario. The software will model the dynamics of a process, through a disturbance, and calculate PID control parameters in response.

Modifications to the PID algorithm

The basic PID algorithm presents some challenges in control applications that have been addressed by minor modifications to the PID form.

Integral windup

One common problem resulting from the ideal PID implementations is integral windup, where a large change in setpoint occurs (say a positive change) and the integral term accumulates an error larger than the maximal value for the the regulation variable (windup), thus the system overshoots and continues to increase as this accumulated error is unwound. This problem can be addressed by:

- Initializing the controller integral to a desired value
- Increasing the setpoint in a suitable ramp
- Disabling the integral function until the PV has entered the controllable region
- Limiting the time period over which the integral error is calculated
- Preventing the integral term from accumulating above or below pre-determined bounds

Overshooting from known disturbances

For example, a PID loop is used to control the temperature of an electric resistance furnace, the system has stabilized. Now the door is opened and something cold is put into the furnace the temperature drops below the setpoint. The integral function of the controller tends to compensate this error by introducing another error in the positive direction. This overshoot can be avoided by freezing of the integral function after the opening of the door for the time the control loop typically needs to reheat the furnace.

Replacing the integral function by a model based part

Often the time-response of the system is approximately known. Then it is an advantage to simulate this time-response with a model and to calculate some unknown parameter from the actual response of the system. If for instance the system is an electrical furnace the response of the difference between furnace temperature and ambient temperature to changes of the electrical power will be similar to that of a simple RC low-pass filter multiplied by an unknown proportional coefficient. The actual electrical power supplied to the furnace is delayed by a low-pass filter to simulate the response of the temperature of the furnace and then the actual temperature minus the ambient temperature is divided by this low-pass filtered electrical power. Then, the result is stabilized by another low-pass filter leading to an estimation of the proportional coefficient. With this estimation, it is possible to calculate the required electrical power by dividing the set-point of the temperature minus the ambient temperature by this coefficient. The result can then be used instead of the integral function. This also achieves a control error of zero in the steady-state, but avoids integral windup and can give a significantly improved control action compared to an optimized PID controller. This type of controller does work properly in an open loop situation which causes integral windup with an integral function. This is an advantage if, for example, the heating of a furnace has to be reduced for some time because of the failure of a heating element, or if the controller is used as an advisory system to a human operator who may not switch it to closed-loop operation. It may also be useful if the controller is inside a branch of a complex control system that may be temporarily inactive.

Many PID loops control a mechanical device (for example, a valve). Mechanical maintenance can be a major cost and wear leads to control degradation in the form of either stiction or a deadband in the mechanical response to an input signal. The rate of mechanical wear is mainly a function of how often a device is activated to make a change. Where wear is a significant concern, the PID loop may have an output deadband to reduce the frequency of activation of the output (valve). This is accomplished by modifying the controller to hold its output steady if the change would be small (within the defined deadband range). The calculated output must leave the deadband before the actual output will change.

The proportional and derivative terms can produce excessive movement in the output when a system is subjected to an instantaneous step increase in the error, such as a large setpoint change. In the case of the derivative term, this is due to taking the derivative of the error, which is very large in the case of an instantaneous step change. As a result, some PID algorithms incorporate the following modifications:

Derivative of output
: In this case the PID controller measures the derivative of the output quantity, rather than the derivative of the error. The output is always continuous (i.e., never has a step change). For this to be effective, the derivative of the output must have the same sign as the derivative of the error.

Setpoint ramping
: In this modification, the setpoint is gradually moved from its old value to a newly specified value using a linear or first order differential ramp function. This avoids the discontinuity present in a simple step change.

Setpoint weighting
: Setpoint weighting uses different multipliers for the error depending on which element of the controller it is used in. The error in the integral term must be the true control error to avoid steady-state control errors. This affects the controller's setpoint response. These parameters do not affect the response to load disturbances and measurement noise.

History

PID theory developed by observing the action of helmsmen.

PID controllers date to 1890s governor design. PID controllers were subsequently developed in automatic ship steering. One of the earliest examples of a PID-type controller was developed by Elmer Sperry in 1911, while the first published theoretical analysis of a PID controller was by Russian American engineer Nicolas Minorsky, in (Minorsky 1922). Minorsky was designing automatic steering systems for the US Navy, and based his analysis on observations of a helmsman, observing that the helmsman controlled the ship not only based on the current error, but also on past error and current rate of change; this was then made mathematical by Minorsky. His goal was stability, not general control, which significantly simplified the problem. While proportional control provides stability against small disturbances, it was insufficient for dealing with a steady disturbance, notably a stiff gale (due to droop), which required adding the integral term. Finally, the derivative term was added to improve control.

Trials were carried out on the USS *New Mexico*, with the controller controlling the *angular velocity* (not angle) of the rudder. PI control yielded sustained yaw (angular error) of ±2°, while adding D yielded yaw of ±1/6°, better than most helmsmen could achieve.

The Navy ultimately did not adopt the system, due to resistance by personnel. Similar work was carried out and published by several others in the 1930s.

Limitations of PID control

While PID controllers are applicable to many control problems, and often perform satisfactorily without any improvements or even tuning, they can perform poorly in some applications, and do not in general provide *optimal* control. The fundamental difficulty with PID control is that it is a feed*back* system, with *constant* parameters, and no direct knowledge of the process, and thus overall performance is reactive and a compromise – while PID control is the best controller with no model of the process, better performance can be obtained by incorporating a model of the process.

The most significant improvement is to incorporate feed-forward control with knowledge about the system, and using the PID only to control error. Alternatively, PIDs can be modified in more minor ways, such as by changing the parameters (either gain scheduling in different use cases or adaptively modifying them based on performance), improving measurement (higher sampling rate, precision, and accuracy, and low-pass filtering if necessary), or cascading multiple PID controllers.

PID controllers, when used alone, can give poor performance when the PID loop gains must be reduced so that the control system does not overshoot, oscillate or *hunt* about the control setpoint value. They also have difficulties in the presence of non-linearities, may trade off regulation versus response time, do not react to changing process behavior (say, the process changes after it has warmed up), and have lag in responding to large disturbances.

Linearity

Another problem faced with PID controllers is that they are linear, and in particular symmetric. Thus, performance of PID controllers in non-linear systems (such as HVAC systems) is variable. For example, in temperature control, a common use case is active heating (via a heating element) but passive cooling (heating off, but no cooling), so overshoot can only be corrected slowly – it cannot be forced downward. In this case the PID should be tuned to be overdamped, to prevent or reduce overshoot, though this reduces performance (it increases settling time).

Noise in derivative

A problem with the derivative term is that small amounts of measurement or process noise can cause large amounts of change in the output. It is often helpful to filter the measurements with a low-pass filter in order to remove higher-frequency noise components. However, low-pass filtering and derivative control can cancel each other out, so reducing noise by instrumentation means is a much better choice. Alternatively, a nonlinear median filter may be used, which improves the filtering efficiency and practical

performance . In some case, the differential band can be turned off in many systems with little loss of control. This is equivalent to using the PID controller as a *PI* controller.

Improvements

Feed-forward

The control system performance can be improved by combining the feedback (or closed-loop) control of a PID controller with feed-forward (or open-loop) control. Knowledge about the system (such as the desired acceleration and inertia) can be fed forward and combined with the PID output to improve the overall system performance. The feed-forward value alone can often provide the major portion of the controller output. The PID controller can be used primarily to respond to whatever difference or *error* remains between the setpoint (SP) and the actual value of the process variable (PV). Since the feed-forward output is not affected by the process feedback, it can never cause the control system to oscillate, thus improving the system response and stability.

For example, in most motion control systems, in order to accelerate a mechanical load under control, more force or torque is required from the prime mover, motor, or actuator. If a velocity loop PID controller is being used to control the speed of the load and command the force or torque being applied by the prime mover, then it is beneficial to take the instantaneous acceleration desired for the load, scale that value appropriately and add it to the output of the PID velocity loop controller. This means that whenever the load is being accelerated or decelerated, a proportional amount of force is commanded from the prime mover regardless of the feedback value. The PID loop in this situation uses the feedback information to change the combined output to reduce the remaining difference between the process setpoint and the feedback value. Working together, the combined open-loop feed-forward controller and closed-loop PID controller can provide a more responsive, stable and reliable control system.

Other improvements

In addition to feed-forward, PID controllers are often enhanced through methods such as PID gain scheduling (changing parameters in different operating conditions), fuzzy logic or computational verb logic . Further practical application issues can arise from instrumentation connected to the controller. A high enough sampling rate, measurement precision, and measurement accuracy are required to achieve adequate control performance.

Cascade control

One distinctive advantage of PID controllers is that two PID controllers can be used together to yield better dynamic performance. This is called cascaded PID control. In cascade control there are two PIDs arranged with one PID controlling the set point of another. A PID controller acts as outer loop controller, which controls the primary

physical parameter, such as fluid level or velocity. The other controller acts as inner loop controller, which reads the output of outer loop controller as set point, usually controlling a more rapid changing parameter, flowrate or acceleration. It can be mathematically proven that the working frequency of the controller is increased and the time constant of the object is reduced by using cascaded PID controller.

Physical implementation of PID control

In the early history of automatic process control the PID controller was implemented as a mechanical device. These mechanical controllers used a lever, spring and a mass and were often energized by compressed air. These pneumatic controllers were once the industry standard.

Electronic analog controllers can be made from a solid-state or tube amplifier, a capacitor and a resistance. Electronic analog PID control loops were often found within more complex electronic systems, for example, the head positioning of a disk drive, the power conditioning of a power supply, or even the movement-detection circuit of a modern seismometer. Nowadays, electronic controllers have largely been replaced by digital controllers implemented with microcontrollers or FPGAs.

Most modern PID controllers in industry are implemented in programmable logic controllers (PLCs) or as a panel-mounted digital controller. Software implementations have the advantages that they are relatively cheap and are flexible with respect to the implementation of the PID algorithm.

Variable voltages may be applied by the time proportioning form of Pulse-width modulation (PWM) – a cycle time is fixed, and variation is achieved by varying the proportion of the time during this cycle that the controller outputs +1 (or −1) instead of 0. On a digital system the possible proportions are discrete – e.g., increments of .1 second within a 2 second cycle time yields 20 possible steps: percentage increments of 5% – so there is a discretization error, but for high enough time resolution this yields satisfactory performance.

Alternative nomenclature and PID forms

Ideal versus standard PID form

The form of the PID controller most often encountered in industry, and the one most relevant to tuning algorithms is the *standard form*. In this form the K_p gain is applied to the I_{out}, and D_{out} terms, yielding:

$$\mathrm{MV(t)} = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau)\,d\tau + T_d \frac{d}{dt} e(t) \right)$$

where

T_i is the *integral time*
T_d is the *derivative time*

In this standard form, the parameters have a clear physical meaning. In particular, the inner summation produces a new single error value which is compensated for future and past errors. The addition of the proportional and derivative components effectively predicts the error value at T_d seconds (or samples) in the future, assuming that the loop control remains unchanged. The integral component adjusts the error value to compensate for the sum of all past errors, with the intention of completely eliminating them in T_i seconds (or samples). The resulting compensated single error value is scaled by the single gain K_p.

In the ideal parallel form, shown in the controller theory section

$$\mathrm{MV(t)} = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{d}{dt} e(t)$$

the gain parameters are related to the parameters of the standard form through $K_i = \frac{K_p}{T_i}$ and $K_d = K_p T_d$. This parallel form, where the parameters are treated as simple gains, is the most general and flexible form. However, it is also the form where the parameters have the least physical interpretation and is generally reserved for theoretical treatment of the PID controller. The standard form, despite being slightly more complex mathematically, is more common in industry.

Laplace form of the PID controller

Sometimes it is useful to write the PID regulator in Laplace transform form:

$$G(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$$

Having the PID controller written in Laplace form and having the transfer function of the controlled system makes it easy to determine the closed-loop transfer function of the system.

PID Pole Zero Cancellation

The PID equation can be written in this form:

$$G(s) = K_d \frac{s^2 + \frac{K_p}{K_d} s + \frac{K_i}{K_d}}{s}$$

When this form is used it is easy to determine the closed loop transfer function.

$$H(s) = \frac{1}{s^2 + 2\zeta\omega_0 s + \omega_0^2}$$

If

$$\frac{K_i}{K_d} = \omega_0^2$$
$$\frac{K_p}{K_d} = 2\zeta\omega_0$$

Then

$$G(s)H(s) = \frac{K_d}{s}$$

This can be very useful to remove unstable poles

Series/interacting form

Another representation of the PID controller is the series, or *interacting* form

$$G(s) = K_c \frac{(\tau_i s + 1)}{\tau_i s}(\tau_d s + 1)$$

where the parameters are related to the parameters of the standard form through

$K_p = K_c \cdot \alpha$, $T_i = \tau_i \cdot \alpha$, and
$T_d = \frac{\tau_d}{\alpha}$

with

$$\alpha = 1 + \frac{\tau_d}{\tau_i}.$$

This form essentially consists of a PD and PI controller in series, and it made early (analog) controllers easier to build. When the controllers later became digital, many kept using the interacting form.

Discrete implementation

The analysis for designing a digital implementation of a PID controller in a Microcontroller (MCU) or FPGA device requires the standard form of the PID controller to be *discretised* . Approximations for first-order derivatives are made by backward finite differences. The integral term is discretised, with a sampling time Δt,as follows,

$$\int_0^{t_k} e(\tau)\,d\tau = \sum_{i=1}^{k} e(t_i)\Delta t$$

The derivative term is approximated as,

$$\frac{de(t_k)}{dt} = \frac{e(t_k) - e(t_{k-1})}{\Delta t}$$

Thus, a *velocity algorithm* for implementation of the discretised PID controller in a MCU is obtained by differentiating $u(t)$, using the numerical definitions of the first and second derivative and solving for $u(t_k)$ and finally obtaining:

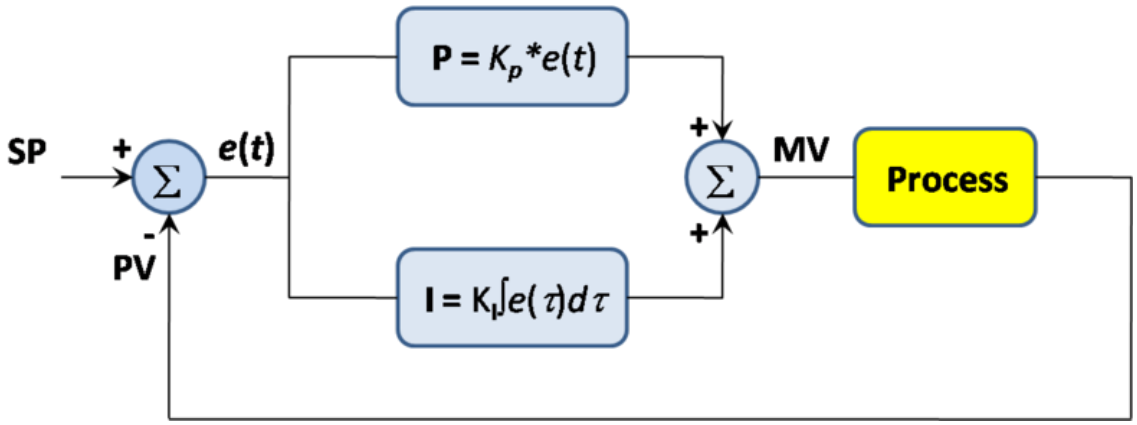
$$u(t_k) = u(t_{k-1}) + K_p \left[\left(1 + \frac{\Delta t}{T_i} + \frac{T_d}{\Delta t}\right) e(t_k) + \left(-1 - \frac{2T_d}{\Delta t}\right) e(t_{k-1}) + \frac{T_d}{\Delta t} e(t_{k-2}) \right]$$

Pseudocode

Here is a simple software loop that implements the PID algorithm in its 'ideal, parallel' form:

```
previous_error = 0
integral = 0
start:
  error = setpoint - actual_position
  integral = integral + (error*dt)
  derivative = (error - previous_error)/dt
  output = (Kp*error) + (Ki*integral) + (Kd*derivative)
  previous_error = error
  wait(dt)
  goto start
```

PI controller



Basic block of a PI controller.

A **PI Controller** (proportional-integral controller) is a special case of the PID controller in which the derivative (D) of the error is not used.

The controller output is given by

$$K_P \Delta + K_I \int \Delta \, dt$$

where Δ is the error or deviation of actual measured value (***PV***) from the set-point (***SP***).

Δ = **SP - PV**.

A PI controller can be modelled easily in software such as Simulink using a "flow chart" box involving Laplace operators:

$$C = \frac{G(1 + \tau s)}{\tau s}$$

where

$G = K_P$ = proportional gain
$G / \tau = K_I$ = integral gain

Setting a value for G is often a trade off between decreasing overshoot and increasing settling time.

The lack of derivative action may make the system more steady in the steady state in the case of noisy data. This is because derivative action is more sensitive to higher-frequency terms in the inputs.

Without derivative action, a PI-controlled system is less responsive to real (non-noise) and relatively fast alterations in state and so the system will be slower to reach setpoint and slower to respond to perturbations than a well-tuned PID system may be.

Chapter- 6

Programmable Logic Controller

Siemens Simatic S7-400 system at rack, left-to-right: power supply unit PS407 4A,CPU 416-3, interface module IM 460-0 and communication processor CP 443-1.

A **programmable logic controller** (**PLC**) or **programmable controller** is a digital computer used for automation of electromechanical processes, such as control of machinery on factory assembly lines, amusement rides, or lighting fixtures. PLCs are used in many industries and machines. Unlike general-purpose computers, the PLC is designed for multiple inputs and output arrangements, extended temperature ranges, immunity to electrical noise, and resistance to vibration and impact. Programs to control machine operation are typically stored in battery-backed or non-volatile memory. A PLC is an example of a *hard* real time system since output results must be produced in response to input conditions within a bounded time, otherwise unintended operation will result.

History

The PLC was invented in response to the needs of the American automotive manufacturing industry. Programmable logic controllers were initially adopted by the automotive industry where software revision replaced the re-wiring of hard-wired control panels when production models changed.

Before the PLC, control, sequencing, and safety interlock logic for manufacturing automobiles was accomplished using hundreds or thousands of relays, cam timers, and drum sequencers and dedicated closed-loop controllers. The process for updating such facilities for the yearly model change-over was very time consuming and expensive, as electricians needed to individually rewire each and every relay.

In 1968 GM Hydramatic (the automatic transmission division of General Motors) issued a request for proposal for an electronic replacement for hard-wired relay systems. The winning proposal came from Bedford Associates of Bedford, Massachusetts. The first PLC, designated the 084 because it was Bedford Associates' eighty-fourth project, was the result. Bedford Associates started a new company dedicated to developing, manufacturing, selling, and servicing this new product: Modicon, which stood for MOdular DIgital CONtroller. One of the people who worked on that project was Dick Morley, who is considered to be the "father" of the PLC. The Modicon brand was sold in 1977 to Gould Electronics, and later acquired by German Company AEG and then by French Schneider Electric, the current owner.

One of the very first 084 models built is now on display at Modicon's headquarters in North Andover, Massachusetts. It was presented to Modicon by GM, when the unit was retired after nearly twenty years of uninterrupted service. Modicon used the 84 moniker at the end of its product range until the 984 made its appearance.

The automotive industry is still one of the largest users of PLCs.

Development

Early PLCs were designed to replace relay logic systems. These PLCs were programmed in "ladder logic", which strongly resembles a schematic diagram of relay logic. This program notation was chosen to reduce training demands for the existing technicians. Other early PLCs used a form of instruction list programming, based on a stack-based logic solver.

Modern PLCs can be programmed in a variety of ways, from ladder logic to more traditional programming languages such as BASIC and C. Another method is State Logic, a very high-level programming language designed to program PLCs based on state transition diagrams.

Many early PLCs did not have accompanying programming terminals that were capable of graphical representation of the logic, and so the logic was instead represented as a series of logic expressions in some version of Boolean format, similar to Boolean algebra. As programming terminals evolved, it became more common for ladder logic to be used, for the aforementioned reasons. Newer formats such as State Logic and Function Block (which is similar to the way logic is depicted when using digital integrated logic circuits) exist, but they are still not as popular as ladder logic. A primary reason for this is that PLCs solve the logic in a predictable and repeating sequence, and ladder logic allows the programmer (the person writing the logic) to see any issues with the timing of the logic sequence more easily than would be possible in other formats.

Programming

Early PLCs, up to the mid-1980s, were programmed using proprietary programming panels or special-purpose programming terminals, which often had dedicated function keys representing the various logical elements of PLC programs. Programs were stored on cassette tape cartridges. Facilities for printing and documentation were very minimal due to lack of memory capacity. The very oldest PLCs used non-volatile magnetic core memory.

More recently, PLCs are programmed using application software on personal computers. The computer is connected to the PLC through Ethernet, RS-232, RS-485 or RS-422 cabling. The programming software allows entry and editing of the ladder-style logic. Generally the software provides functions for debugging and troubleshooting the PLC software, for example, by highlighting portions of the logic to show current status during operation or via simulation. The software will upload and download the PLC program, for backup and restoration purposes. In some models of programmable controller, the program is transferred from a personal computer to the PLC though a programming board which writes the program into a removable chip such as an EEPROM or EPROM.

Functionality

The functionality of the PLC has evolved over the years to include sequential relay control, motion control, process control, distributed control systems and networking. The data handling, storage, processing power and communication capabilities of some modern PLCs are approximately equivalent to desktop computers. PLC-like programming combined with remote I/O hardware, allow a general-purpose desktop computer to overlap some PLCs in certain applications. Regarding the practicality of these desktop computer based logic controllers, it is important to note that they have not been generally accepted in heavy industry because the desktop computers run on less stable operating systems than do PLCs, and because the desktop computer hardware is typically not designed to the same levels of tolerance to temperature, humidity, vibration, and longevity as the processors used in PLCs. In addition to the hardware limitations of desktop based logic, operating systems such as Windows do not lend themselves to deterministic logic execution, with the result that the logic may not always respond to changes in logic state or input status with the extreme consistency in timing as is expected from PLCs. Still, such desktop logic applications find use in less critical situations, such as laboratory automation and use in small facilities where the application is less demanding and critical, because they are generally much less expensive than PLCs.

In more recent years, small products called PLRs (programmable logic relays), and also by similar names, have become more common and accepted. These are very much like PLCs, and are used in light industry where only a few points of I/O (i.e. a few signals coming in from the real world and a few going out) are involved, and low cost is desired. These small devices are typically made in a common physical size and shape by several manufacturers, and branded by the makers of larger PLCs to fill out their low end product range. Popular names include PICO Controller, NANO PLC, and other names implying very small controllers. Most of these have between 8 and 12 digital inputs, 4 and 8 digital outputs, and up to 2 analog inputs. Size is usually about 4" wide, 3" high, and 3" deep. Most such devices include a tiny postage stamp sized LCD screen for viewing simplified ladder logic (only a very small portion of the program being visible at a given time) and status of I/O points, and typically these screens are accompanied by a 4-way rocker push-button plus four more separate push-buttons, similar to the key buttons on a VCR remote control, and used to navigate and edit the logic. Most have a small plug for connecting via RS-232 or RS-485 to a personal computer so that programmers can use simple Windows applications for programming instead of being forced to use the tiny LCD and push-button set for this purpose. Unlike regular PLCs that are usually modular and greatly expandable, the PLRs are usually not modular or expandable, but their price can be two orders of magnitude less than a PLC and they still offer robust design and deterministic execution of the logic.

PLC Topics

Features

Control panel with PLC (grey elements in the center). The unit consists of separate elements, from left to right; power supply, controller, relay units for in- and output

The main difference from other computers is that PLCs are armored for severe conditions (such as dust, moisture, heat, cold) and have the facility for extensive input/output (I/O) arrangements. These connect the PLC to sensors and actuators. PLCs read limit switches, analog process variables (such as temperature and pressure), and the positions of complex positioning systems. Some use machine vision. On the actuator side, PLCs operate electric motors, pneumatic or hydraulic cylinders, magnetic relays, solenoids, or analog

outputs. The input/output arrangements may be built into a simple PLC, or the PLC may have external I/O modules attached to a computer network that plugs into the PLC.

System scale

A small PLC will have a fixed number of connections built in for inputs and outputs. Typically, expansions are available if the base model has insufficient I/O.

Modular PLCs have a chassis (also called a rack) into which are placed modules with different functions. The processor and selection of I/O modules is customised for the particular application. Several racks can be administered by a single processor, and may have thousands of inputs and outputs. A special high speed serial I/O link is used so that racks can be distributed away from the processor, reducing the wiring costs for large plants.

User interface

PLCs may need to interact with people for the purpose of configuration, alarm reporting or everyday control.

A Human-Machine Interface (HMI) is employed for this purpose. HMIs are also referred to as MMIs (Man Machine Interface) and GUIs (Graphical User Interface).

A simple system may use buttons and lights to interact with the user. Text displays are available as well as graphical touch screens. More complex systems use programming and monitoring software installed on a computer, with the PLC connected via a communication interface.

Communications

PLCs have built in communications ports, usually 9-pin RS-232, but optionally EIA-485 or Ethernet. Modbus, BACnet or DF1 is usually included as one of the communications protocols. Other options include various fieldbuses such as DeviceNet or Profibus. Other communications protocols that may be used are listed in the List of automation protocols.

Most modern PLCs can communicate over a network to some other system, such as a computer running a SCADA (Supervisory Control And Data Acquisition) system or web browser.

PLCs used in larger I/O systems may have peer-to-peer (P2P) communication between processors. This allows separate parts of a complex process to have individual control while allowing the subsystems to co-ordinate over the communication link. These communication links are also often used for HMI devices such as keypads or PC-type workstations.

Programming

PLC programs are typically written in a special application on a personal computer, then downloaded by a direct-connection cable or over a network to the PLC. The program is stored in the PLC either in battery-backed-up RAM or some other non-volatile flash memory. Often, a single PLC can be programmed to replace thousands of relays.

Under the IEC 61131-3 standard, PLCs can be programmed using standards-based programming languages. A graphical programming notation called Sequential Function Charts is available on certain programmable controllers. Initially most PLCs utilized Ladder Logic Diagram Programming, a model which emulated electromechanical control panel devices (such as the contact and coils of relays) which PLCs replaced. This model remains common today.

IEC 61131-3 currently defines five programming languages for programmable control systems: FBD (Function block diagram), LD (Ladder diagram), ST (Structured text, similar to the Pascal programming language), IL (Instruction list, similar to assembly language) and SFC (Sequential function chart). These techniques emphasize logical organization of operations.

While the fundamental concepts of PLC programming are common to all manufacturers, differences in I/O addressing, memory organization and instruction sets mean that PLC programs are never perfectly interchangeable between different makers. Even within the same product line of a single manufacturer, different models may not be directly compatible.

PLC compared with other control systems

Allen-Bradley PLC installed in a control panel

PLCs are well-adapted to a range of automation tasks. These are typically industrial processes in manufacturing where the cost of developing and maintaining the automation system is high relative to the total cost of the automation, and where changes to the system would be expected during its operational life. PLCs contain input and output devices compatible with industrial pilot devices and controls; little electrical design is required, and the design problem centers on expressing the desired sequence of operations. PLC applications are typically highly customized systems so the cost of a packaged PLC is low compared to the cost of a specific custom-built controller design. On the other hand, in the case of mass-produced goods, customized control systems are economic due to the lower cost of the components, which can be optimally chosen instead of a "generic" solution, and where the non-recurring engineering charges are spread over thousands or millions of units.

For high volume or very simple fixed automation tasks, different techniques are used. For example, a consumer dishwasher would be controlled by an electromechanical cam timer costing only a few dollars in production quantities.

A microcontroller-based design would be appropriate where hundreds or thousands of units will be produced and so the development cost (design of power supplies, input/output hardware and necessary testing and certification) can be spread over many sales, and where the end-user would not need to alter the control. Automotive

applications are an example; millions of units are built each year, and very few end-users alter the programming of these controllers. However, some specialty vehicles such as transit busses economically use PLCs instead of custom-designed controls, because the volumes are low and the development cost would be uneconomic.

Very complex process control, such as used in the chemical industry, may require algorithms and performance beyond the capability of even high-performance PLCs. Very high-speed or precision controls may also require customized solutions; for example, aircraft flight controls.

Programmable controllers are widely used in motion control, positioning control and torque control. Some manufacturers produce motion control units to be integrated with PLC so that G-code (involving a CNC machine) can be used to instruct machine movements.

PLCs may include logic for single-variable feedback analog control loop, a "proportional, integral, derivative" or "PID controller". A PID loop could be used to control the temperature of a manufacturing process, for example. Historically PLCs were usually configured with only a few analog control loops; where processes required hundreds or thousands of loops, a distributed control system (DCS) would instead be used. As PLCs have become more powerful, the boundary between DCS and PLC applications has become less distinct.

PLCs have similar functionality as Remote Terminal Units. An RTU, however, usually does not support control algorithms or control loops. As hardware rapidly becomes more powerful and cheaper, RTUs, PLCs and DCSs are increasingly beginning to overlap in responsibilities, and many vendors sell RTUs with PLC-like features and vice versa. The industry has standardized on the IEC 61131-3 functional block language for creating programs to run on RTUs and PLCs, although nearly all vendors also offer proprietary alternatives and associated development environments.

Digital and analog signals

Digital or discrete signals behave as binary switches, yielding simply an On or Off signal (1 or 0, True or False, respectively). Push buttons, limit switches, and photoelectric sensors are examples of devices providing a discrete signal. Discrete signals are sent using either voltage or current, where a specific range is designated as *On* and another as *Off*. For example, a PLC might use 24 V DC I/O, with values above 22 V DC representing *On*, values below 2VDC representing *Off*, and intermediate values undefined. Initially, PLCs had only discrete I/O.

Analog signals are like volume controls, with a range of values between zero and full-scale. These are typically interpreted as integer values (counts) by the PLC, with various ranges of accuracy depending on the device and the number of bits available to store the data. As PLCs typically use 16-bit signed binary processors, the integer values are limited between -32,768 and +32,767. Pressure, temperature, flow, and weight are often

represented by analog signals. Analog signals can use voltage or current with a magnitude proportional to the value of the process signal. For example, an analog 0 - 10 V input or 4-20 mA would be converted into an integer value of 0 - 32767.

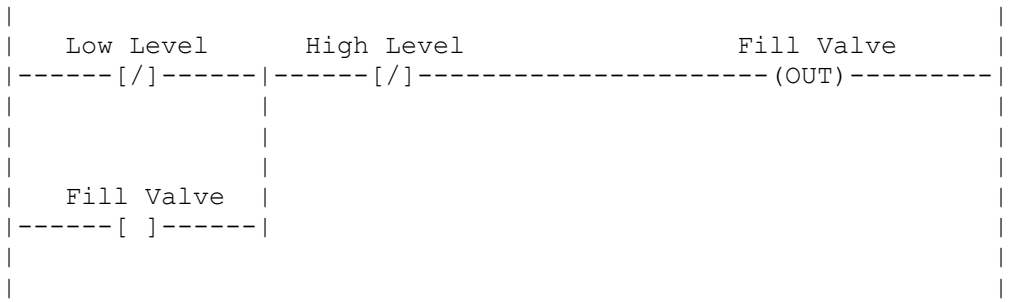
Current inputs are less sensitive to electrical noise (i.e. from welders or electric motor starts) than voltage inputs.

Example

As an example, say a facility needs to store water in a tank. The water is drawn from the tank by another system, as needed, and our example system must manage the water level in the tank.

Using only digital signals, the PLC has two digital inputs from float switches (Low Level and High Level). When the water level is above the switch it closes a contact and passes a signal to an input. The PLC uses a digital output to open and close the inlet valve into the tank.

When the water level drops enough so that the Low Level float switch is off (down), the PLC will open the valve to let more water in. Once the water level rises enough so that the High Level switch is on (up), the PLC will shut the inlet to stop the water from overflowing. This rung is an example of seal-in (latching) logic. The output is sealed in until some condition breaks the circuit.



An analog system might use a water pressure sensor or a load cell, and an adjustable (throttling) dripping out of the tank, the valve adjusts to slowly drip water back into the tank.

In this system, to avoid 'flutter' adjustments that can wear out the valve, many PLCs incorporate "hysteresis" which essentially creates a "deadband" of activity. A technician adjusts this deadband so the valve moves only for a significant change in rate. This will in turn minimize the motion of the valve, and reduce its wear.

A real system might combine both approaches, using float switches and simple valves to prevent spills, and a rate sensor and rate valve to optimize refill rates and prevent water hammer. Backup and maintenance methods can make a real system very complicated.

Chapter- 7

State Observer

In control theory, a **state observer** is a system that models a real system in order to provide an estimate of its internal state, given measurements of the input and output of the real system. It is typically a computer-implemented mathematical model.

Knowing the system state is necessary to solve many control theory problems; for example, stabilizing a system using state feedback. In most practical cases, the physical state of the system cannot be determined by direct observation. Instead, indirect effects of the internal state are observed by way of the system outputs. A simple example is that of vehicles in a tunnel: the rates and velocities at which vehicles enter and leave the tunnel can be observed directly, but the exact state inside the tunnel can only be estimated. If a system is observable, it is possible to fully reconstruct the system state from its output measurements using the state observer.

Typical observer model

The state of a physical discrete-time system is assumed to satisfy

$$\begin{aligned}\mathbf{x}(k+1) &= A\mathbf{x}(k) + B\mathbf{u}(k)\\ \mathbf{y}(k) &= C\mathbf{x}(k) + D\mathbf{u}(k)\end{aligned}$$

where, at time *k*, $\mathbf{x}(k)$is the plant's state; $\mathbf{u}(k)$is its inputs; and $\mathbf{y}(k)$is its outputs. These equations simply say that the plant's current outputs and its future state are both determined solely by its current state and the current inputs. (Although these equations are expressed in terms of discrete time steps, very similar equations hold for continuous systems). If this system is observable then the output of the plant, $\mathbf{y}(k)$, can be used to steer the state of the state observer.

The observer model of the physical system is then typically derived from the above equations. Additional terms may be included in order to ensure that, on receiving

successive measured values of the plant's inputs and outputs, the model's state converges to that of the plant. In particular, the output of the observer may be subtracted from the output of the plant and then multiplied by a matrix *L*; this is then added to the equations for the state of the observer to produce a so-called *Luenberger observer*, defined by the equations below. Note that the variables of a state observer are commonly denoted by a "hat": $\hat{\mathbf{x}}(k)$ and $\hat{\mathbf{y}}(k)$ to distinguish them from the variables of the equations satisfied by the physical system.

$$\begin{aligned}\hat{\mathbf{x}}(k+1) &= A\hat{\mathbf{x}}(k) + L\left[\mathbf{y}(k) - \hat{\mathbf{y}}(k)\right] + B\mathbf{u}(k)\\ \hat{\mathbf{y}}(k) &= C\hat{\mathbf{x}}(k) + D\mathbf{u}(k)\end{aligned}$$

The observer is called asymptotically stable if the observer error $\mathbf{e}(k) = \hat{\mathbf{x}}(k) - \mathbf{x}(k)$ converges to zero when $k \to \infty$. For a Luenberger observer, the observer error satisfies $\mathbf{e}(k+1) = (A - LC)\mathbf{e}(k)$. The Luenberger observer for this discrete-time system is therefore asymptotically stable when the matrix $A - LC$ has all the eigenvalues inside the unit circle.

For control purposes the output of the observer system is fed back to the input of both the observer and the plant through the gains matrix *K*.

$$\mathbf{u}(\mathbf{k}) = -K\hat{\mathbf{x}}(k)$$

The observer equations then become:

$$\begin{aligned}\hat{\mathbf{x}}(k+1) &= A\hat{\mathbf{x}}(k) + L\left(\mathbf{y}(k) - \hat{\mathbf{y}}(k)\right) - BK\hat{\mathbf{x}}(k)\\ \hat{\mathbf{y}}(k) &= C\hat{\mathbf{x}}(k) - DK\hat{\mathbf{x}}(k)\end{aligned}$$

or, more simply,

$$\begin{aligned}\hat{\mathbf{x}}(k+1) &= (A - BK)\,\hat{\mathbf{x}}(k) + L\left(\mathbf{y}(k) - \hat{\mathbf{y}}(k)\right)\\ \hat{\mathbf{y}}(k) &= (C - DK)\,\hat{\mathbf{x}}(k)\end{aligned}$$

Due to the separation principle we know that we can choose *K* and *L* independently without harm to the overall stability of the systems. As a rule of thumb, the poles of the observer $A - LC$ are usually chosen to converge 10 times faster than the poles of the system $A - BK$.

Continuous-time case

The previous example was for an observer implemented in a discrete-time LTI system. However, the process is similar for the continuous-time case; the observer gains *L* are chosen to make the continuous-time error dynamics converge to zero asymptotically (i.e., when $A - LC$ is a Hurwitz matrix).

For a continuous-time linear system

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u},$$
$$\mathbf{y} = C\mathbf{x},$$

where $\mathbf{x} \in \mathbb{R}^n, \mathbf{u} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^r$, the observer looks similar to discrete-time case described above:

$$\dot{\hat{\mathbf{x}}} = A\hat{\mathbf{x}} + B\mathbf{u} + L\left(\mathbf{y} - C\hat{\mathbf{x}}\right).$$

The observer error $\mathbf{e} = \hat{\mathbf{x}} - \mathbf{x}$satisfies the equation

$$\dot{\mathbf{e}} = (A - LC)\mathbf{e}.$$

The eigenvalues of the matrix $A - LC$ can be made arbitrarily by appropriate choice of the observer gain L when the pair $[A,C]$ is observable, i.e. observability condition holds. In particular, it can be made Hurwitz, so the observer error $e(t) \to 0$when $t \to \infty$.

Peaking and other observer methods

When the observer gain L is high, the linear Luenberger observer converges to the system states very quickly. However, high observer gain leads to a peaking phenomenon in which initial estimator error can be prohibitively large (i.e., impractical or unsafe to use). As a consequence, nonlinear high gain observer methods are available that converge quickly without the peaking phenomenon. For example, sliding mode control can be used to design an observer that brings one estimated state's error to zero in finite time even in the presence of measurement error; the other states have error that behaves similarly to the error in a Luenberger observer after peaking has subsided. Sliding mode observers also have attractive noise resilience properties that are similar to a Kalman filter.

State observers for nonlinear systems

Sliding mode observers can be designed for the non-linear systems as well. For simplicity, first consider the no-input non-linear system:

$$\dot{\mathbf{x}} = f(\mathbf{x})$$

where $\mathbf{x} \in \mathbb{R}^n$. Also assume that there is a measurable output $\mathbf{y} \in \mathbb{R}$given by

$$\mathbf{y} = h(\mathbf{x}).$$

There are several non-approximate approaches for designing an observer. The two observers given below also apply to the case when the system has an input. That is,

$$\dot{\mathbf{x}} = f(\mathbf{x}) + B(\mathbf{x})\mathbf{u},$$

$$\mathbf{y} = h(\mathbf{x}),$$

Linearizable error dynamics

One suggested by Kerner and Isidori and Krener and Respondek can be applied in a situation when there exists a linearizing transformation (i.e., a diffeomorphism, like the one used in feedback linearization) $\mathbf{z} = \Phi(\mathbf{x})$ such that in new variables the system equations read

$$\begin{aligned}\dot{\mathbf{z}} &= A\mathbf{z} + \phi(\mathbf{y}),\\ \mathbf{y} &= C\mathbf{z}.\end{aligned}$$

The Luenberger observer is then designed as

$$\dot{\hat{\mathbf{z}}} = A\hat{\mathbf{z}} + \phi(\mathbf{y}) - L\left(C\hat{\mathbf{z}} - \mathbf{y}\right).$$

The observer error for the transformed variable $\mathbf{e} = \hat{\mathbf{z}} - \mathbf{z}$ satisfies the same equation as in classical linear case.

$$\dot{\mathbf{e}} = (A - LC)\mathbf{e}.$$

As shown by Gauthier, Hammouri, and Othman and Hammouri and Kinnaert, if there exists transformation $\mathbf{z} = \Phi(\mathbf{x})$ such that the system can be transformed into the form

$$\begin{aligned}\dot{\mathbf{z}} &= A(u(t))\mathbf{z} + \phi(\mathbf{y}, u(t)),\\ \mathbf{y} &= C\mathbf{z},\end{aligned}$$

then the observer is designed as

$$\dot{\hat{\mathbf{z}}} = A(u(t))\hat{\mathbf{z}} + \phi(\mathbf{y}, u(t)) - L(t)\left(C\hat{\mathbf{z}} - \mathbf{y}\right),$$

where $L(t)$ is a time-varying observer gain.

Sliding mode observer

As discussed for the linear case above, the peaking phenomenon present in Luenberger observers justifies the use of a sliding mode observer. The sliding mode observer uses non-linear high-gain feedback to drive estimated states to a hypersurface where there is no difference between the estimated output and the measured output. The non-linear gain used in the observer is typically implemented with a scaled switching function, like the signum (i.e., sgn) of the estimated–measured output error. Hence, due to this high-gain feedback, the vector field of the observer has a crease in it so that observer trajectories *slide along* a curve where the estimated output matches the measured output exactly. So,

if the system is observable from its output, the observer states will all be driven to the actual system states. Additionally, by using the sign of the error to drive the sliding mode observer, the observer trajectories become insensitive to many forms of noise. Hence, some sliding mode observers have attractive properties similar to the Kalman filter but with simpler implementation.

As suggested by Drakunov, a sliding mode observer can also be designed for a class of non-linear systems. Such an observer can be written in terms of original variable estimate $\hat{\mathbf{x}}$ and has the form

$$\dot{\hat{\mathbf{x}}} = \left[\frac{\partial H(\hat{\mathbf{x}})}{\partial \mathbf{x}}\right]^{-1} M(\hat{\mathbf{x}})\,\mathrm{sgn}(V(t) - H(\hat{\mathbf{x}}))$$

where:

- The $\mathrm{sgn}(\cdot)$ vector extends the scalar signum function to *n* dimensions. That is,

$$\mathrm{sgn}(\mathbf{z}) = \begin{bmatrix} \mathrm{sgn}(z_1) \\ \mathrm{sgn}(z_2) \\ \vdots \\ \mathrm{sgn}(z_i) \\ \vdots \\ \mathrm{sgn}(z_n) \end{bmatrix}$$

  for the vector $\mathbf{z} \in \mathbb{R}^n$.

- The vector $H(\mathbf{x})$ has components that are the output function $h(\mathbf{x})$ and its repeated Lie derivatives. In particular,

$$H(\mathbf{x}) \triangleq \begin{bmatrix} h_1(\mathbf{x}) \\ h_2(\mathbf{x}) \\ h_3(\mathbf{x}) \\ \vdots \\ h_n(\mathbf{x}) \end{bmatrix} \triangleq \begin{bmatrix} h(\mathbf{x}) \\ L_f h(\mathbf{x}) \\ L_f^2 h(\mathbf{x}) \\ \vdots \\ L_f^{n-1} h(\mathbf{x}) \end{bmatrix}$$

  where $L_f^i h$ is the i^{th} Lie derivative of output function *h* along the vector field *f* (i.e., along $\mathbf{x}$ trajectories of the non-linear system). In the special case where the system has no input or has a relative degree of *n*, $H(\mathbf{x}(t))$ is a collection of the output $\mathbf{y}(t) = h(\mathbf{x}(t))$ and its $n - 1$ derivatives. Because the inverse of the

Jacobian linearization of $H(\mathbf{x})$ must exist for this observer to be well defined, the transformation $H(\mathbf{x})$ is guaranteed to be a local diffeomorphism.

- The diagonal matrix $M(\hat{\mathbf{x}})$ of gains is such that

$$M(\hat{\mathbf{x}}) \triangleq \operatorname{diag}(m_1(\hat{\mathbf{x}}), m_2(\hat{\mathbf{x}}), \ldots, m_n(\hat{\mathbf{x}})) = \begin{bmatrix} m_1(\hat{\mathbf{x}}) & & & & & \\ & m_2(\hat{\mathbf{x}}) & & & & \\ & & \ddots & & & \\ & & & m_i(\hat{\mathbf{x}}) & & \\ & & & & \ddots & \\ & & & & & m_n(\hat{\mathbf{x}}) \end{bmatrix}$$

where, for each $i \in \{1, 2, \ldots, n\}$, element $m_i(\hat{\mathbf{x}}) > 0$ and suitably large to ensure reachability of the sliding mode.

- The observer vector $V(t)$ is such that

$$V(t) \triangleq \begin{bmatrix} v_1(t) \\ v_2(t) \\ v_3(t) \\ \vdots \\ v_i(t) \\ \vdots \\ v_n(t) \end{bmatrix} \triangleq \begin{bmatrix} \mathbf{y}(t) \\ \{m_1(\hat{\mathbf{x}})\operatorname{sgn}(v_1(t) - h_1(\hat{\mathbf{x}}(t)))\}_{\text{eq}} \\ \{m_2(\hat{\mathbf{x}})\operatorname{sgn}(v_2(t) - h_2(\hat{\mathbf{x}}(t)))\}_{\text{eq}} \\ \vdots \\ \{m_{i-1}(\hat{\mathbf{x}})\operatorname{sgn}(v_{i-1}(t) - h_{i-1}(\hat{\mathbf{x}}(t)))\}_{\text{eq}} \\ \vdots \\ \{m_{n-1}(\hat{\mathbf{x}})\operatorname{sgn}(v_{n-1}(t) - h_{n-1}(\hat{\mathbf{x}}(t)))\}_{\text{eq}} \end{bmatrix}$$

where $\operatorname{sgn}(\cdot)$ here is the normal signum function defined for scalars, and $\{\cdots\}_{\text{eq}}$ denotes an "equivalent value operator" of a discontinuous function in sliding mode.

The idea can be briefly explained as follows. According to the theory of sliding modes, in order to describe the system behavior, once sliding mode starts, the function $\operatorname{sgn}(v_i(t) - h_i(\hat{\mathbf{x}}(t)))$ should be replaced by equivalent values. In practice, it switches (chatters) with high frequency with slow component being equal to the equivalent value. Applying appropriate lowpass filter to get rid of the high frequency component on can obtain the value of the equivalent control, which contains more information about the state of the estimated system. The observer described above uses this method several times to obtain the state of the nonlinear system ideally in finite time.

The modified observation error can be written in the transformed states $\mathbf{e} = H(\mathbf{x}) - H(\hat{\mathbf{x}})$. In particular,

$$\begin{aligned}\dot{\mathbf{e}} &= \frac{\mathrm{d}}{\mathrm{d}t}H(\mathbf{x}) - \frac{\mathrm{d}}{\mathrm{d}t}H(\hat{\mathbf{x}}) \\ &= \frac{\mathrm{d}}{\mathrm{d}t}H(\mathbf{x}) - M(\hat{\mathbf{x}})\,\mathrm{sgn}(V(t) - H(\hat{\mathbf{x}}(t))),\end{aligned}$$

and so

$$\begin{bmatrix}\dot{\mathbf{e}}_1\\ \dot{\mathbf{e}}_2\\ \vdots\\ \dot{\mathbf{e}}_i\\ \vdots\\ \dot{\mathbf{e}}_{n-1}\\ \dot{\mathbf{e}}_n\end{bmatrix} = \overbrace{\begin{bmatrix}\dot{h}_1(\mathbf{x})\\ \dot{h}_2(\mathbf{x})\\ \vdots\\ \dot{h}_i(\mathbf{x})\\ \vdots\\ \dot{h}_{n-1}(\mathbf{x})\\ \dot{h}_n(\mathbf{x})\end{bmatrix}}^{\frac{\mathrm{d}}{\mathrm{d}t}H(\mathbf{x})} - \overbrace{M(\hat{\mathbf{x}})\,\mathrm{sgn}(V(t) - H(\hat{\mathbf{x}}(t)))}^{\frac{\mathrm{d}}{\mathrm{d}t}H(\hat{\mathbf{x}})} = \begin{bmatrix}h_2(\mathbf{x})\\ h_3(\mathbf{x})\\ \vdots\\ h_{i+1}(\mathbf{x})\\ \vdots\\ h_n(\mathbf{x})\\ L_f^n h(\mathbf{x})\end{bmatrix} - \begin{bmatrix}m_1\mathrm{sgn}(v_1(t) - h_1(\hat{\mathbf{x}}(t)))\\ m_2\mathrm{sgn}(v_2(t) - h_2(\hat{\mathbf{x}}(t)))\\ \vdots\\ m_i\mathrm{sgn}(v_i(t) - h_i(\hat{\mathbf{x}}(t)))\\ \vdots\\ m_{n-1}\mathrm{sgn}(v_{n-1}(t) - h_{n-1}(\hat{\mathbf{x}}(t)))\\ m_n\mathrm{sgn}(v_n(t) - h_n(\hat{\mathbf{x}}(t)))\end{bmatrix}$$

$$= \begin{bmatrix}\overbrace{v_1(t) = y(t) = h_1(\mathbf{x})}^{\mathbf{e}_1}\\ h_2(\mathbf{x}) - m_1(\hat{\mathbf{x}})\mathrm{sgn}(\overbrace{v_1(t)}\ - h_1(\hat{\mathbf{x}}(t)))\\ h_3(\mathbf{x}) - m_2(\hat{\mathbf{x}})\mathrm{sgn}(v_2(t) - h_2(\hat{\mathbf{x}}(t)))\\ \vdots\\ h_{i+1}(\mathbf{x}) - m_i(\hat{\mathbf{x}})\mathrm{sgn}(v_i(t) - h_i(\hat{\mathbf{x}}(t)))\\ \vdots\\ h_n(\mathbf{x}) - m_{n-1}(\hat{\mathbf{x}})\mathrm{sgn}(v_{n-1}(t) - h_{n-1}(\hat{\mathbf{x}}(t)))\\ L_f^n h(\mathbf{x}) - m_n(\hat{\mathbf{x}})\mathrm{sgn}(v_n(t) - h_n(\hat{\mathbf{x}}(t)))\end{bmatrix}.$$

So:

1. As long as $m_1(\hat{\mathbf{x}}) \geq |h_2(\mathbf{x}(t))|$, the first row of the error dynamics, $\dot{\mathbf{e}}_1 = h_2(\hat{\mathbf{x}}) - m_1(\hat{\mathbf{x}})\mathrm{sgn}(\mathbf{e}_1)$, will meet sufficient conditions to enter the $e_1 = 0$ sliding mode in finite time.
2. Along the $e_1 = 0$ surface, the corresponding $v_2(t) = \{m_1(\hat{\mathbf{x}})\mathrm{sgn}(\mathbf{e}_1)\}_{\mathrm{eq}}$equivalent control will be equal to $h_2(\mathbf{x})$, and so $v_2(t) - h_2(\hat{\mathbf{x}}) = h_2(\mathbf{x}) - h_2(\hat{\mathbf{x}}) = \mathbf{e}_2$. Hence, so long as $m_2(\hat{\mathbf{x}}) \geq |h_3(\mathbf{x}(t))|$, the second row of the error dynamics, $\dot{\mathbf{e}}_2 = h_3(\hat{\mathbf{x}}) - m_2(\hat{\mathbf{x}})\mathrm{sgn}(\mathbf{e}_2)$, will enter the $e_2 = 0$ sliding mode in finite time.
3. Along the $e_i = 0$ surface, the corresponding $v_{i+1}(t) = \{\ldots\}_{\mathrm{eq}}$equivalent control will be equal to $h_{i+1}(\mathbf{x})$. Hence, so long as $m_{i+1}(\hat{\mathbf{x}}) \geq |h_{i+2}(\mathbf{x}(t))|$, the $(i+1)^{\mathrm{th}}$ row of the error dynamics, $\dot{\mathbf{e}}_{i+1} = h_{i+2}(\hat{\mathbf{x}}) - m_{i+1}(\hat{\mathbf{x}})\mathrm{sgn}(\mathbf{e}_{i+1})$, will enter the $e_{i+1} = 0$ sliding mode in finite time.

So, for sufficiently large m_i gains, all observer estimated states reach the actual states in finite time. In fact, increasing m_i allows for convergence in any desired finite time so long as each $|h_i(\mathbf{x}(0))|$function can be bounded with certainty. Hence, the requirement that the map $H : \mathbb{R}^n \to \mathbb{R}^n$is a diffeomorphism (i.e., that its Jacobian linearization is invertible) asserts that convergence of the estimated output implies convergence of the estimated state. That is, the requirement is an observability condition.

In the case of the sliding mode observer for the system with the input, additional conditions are needed for the observation error to be independent of the input. For example, that

$$\frac{\partial H(\mathbf{x})}{\partial \mathbf{x}} B(\mathbf{x})$$

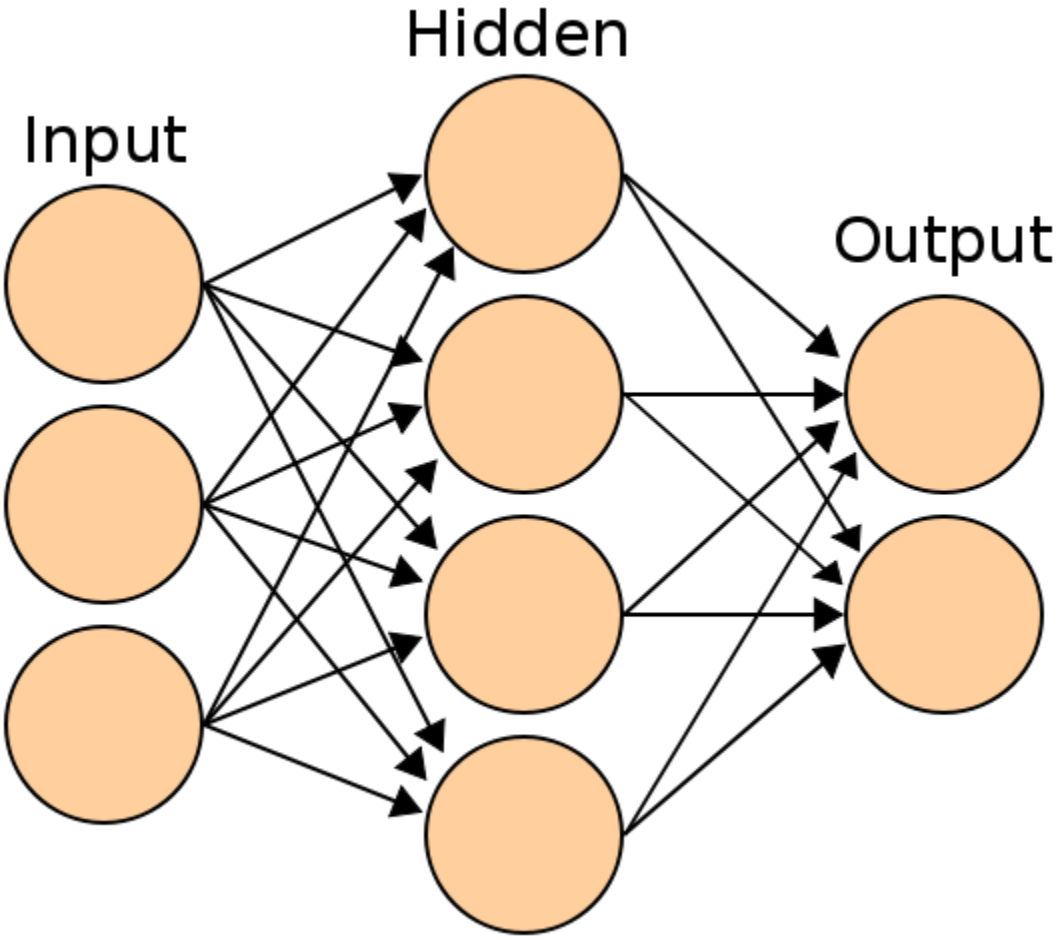
does not depend on time. The observer is then

$$\dot{\hat{x}} = \left[\frac{\partial H(\hat{\mathbf{x}})}{\partial \mathbf{x}}\right]^{-1} M(\hat{\mathbf{x}}) \mathrm{sgn}(V(t) - H(\hat{\mathbf{x}})) + B(\hat{\mathbf{x}})u.$$

Chapter- 8

Artificial Neural Network

An **artificial neural network (ANN)**, usually called **neural network (NN)**, is a mathematical model or computational model that is inspired by the structure and/or functional aspects of biological neural networks. A neural network consists of an interconnected group of artificial neurons, and it processes information using a connectionist approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs or to find patterns in data.



An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in the human brain.

Background

The original inspiration for the term *Artificial Neural Network* came from examination of central nervous systems and their neurons, axons, dendrites, and synapses, which constitute the processing elements of biological neural networks investigated by neuroscience. In an artificial neural network, simple artificial nodes, variously called "neurons", "neurodes", "processing elements" (PEs) or "units", are connected together to form a network of nodes mimicking the biological neural networks — hence the term "artificial neural network".

Because neuroscience is still full of unanswered questions, and since there are many levels of abstraction and therefore many ways to take inspiration from the brain, there is no single formal definition of what an artificial neural network is. Generally, it involves a network of simple processing elements that exhibit complex global behavior determined by connections between processing elements and element parameters. While an artificial

neural network does not have to be adaptive per se, its practical use comes with algorithms designed to alter the strength (weights) of the connections in the network to produce a desired signal flow.

These networks are also similar to the biological neural networks in the sense that functions are performed collectively and in parallel by the units, rather than there being a clear delineation of subtasks to which various units are assigned. Currently, the term Artificial Neural Network (ANN) tends to refer mostly to neural network models employed in statistics, cognitive psychology and artificial intelligence. Neural network models designed with emulation of the central nervous system (CNS) in mind are a subject of theoretical neuroscience and computational neuroscience.

In modern software implementations of artificial neural networks, the approach inspired by biology has been largely abandoned for a more practical approach based on statistics and signal processing. In some of these systems, neural networks or parts of neural networks (such as artificial neurons) are used as components in larger systems that combine both adaptive and non-adaptive elements. While the more general approach of such adaptive systems is more suitable for real-world problem solving, it has far less to do with the traditional artificial intelligence connectionist models. What they do have in common, however, is the principle of non-linear, distributed, parallel and local processing and adaptation.

Models

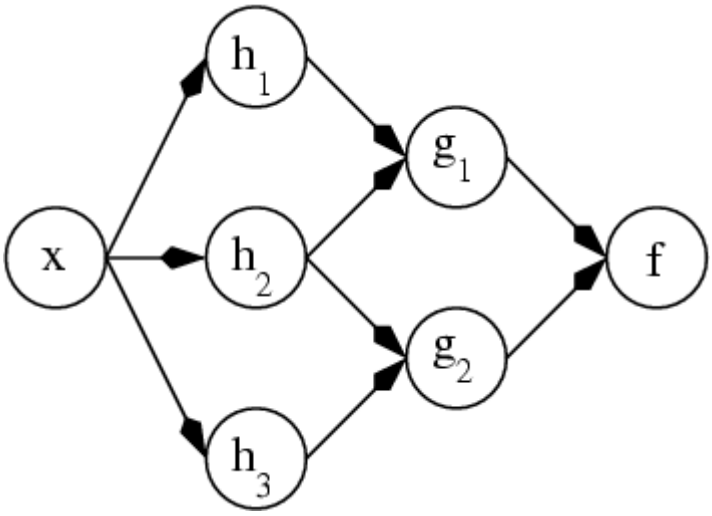
Neural network models in artificial intelligence are usually referred to as artificial neural networks (ANNs); these are essentially simple mathematical models defining a function $f : X \rightarrow Y$ or a distribution over X or both X and Y, but sometimes models are also intimately associated with a particular learning algorithm or learning rule. A common use of the phrase ANN model really means the definition of a *class* of such functions (where members of the class are obtained by varying parameters, connection weights, or specifics of the architecture such as the number of neurons or their connectivity).

Network function

The word *network* in the term 'artificial neural network' refers to the inter–connections between the neurons in the different layers of each system. The most basic system has three layers. The first layer has input neurons, which send data via synapses to the second layer of neurons, and then via more synapses to the third layer of output neurons. More complex systems will have more layers of neurons with some having increased layers of input neurons and output neurons. The synapses store parameters called "weights" that manipulate the data in the calculations.

The layers network through the mathematics of the system algorithms. The network function $f(x)$ is defined as a composition of other functions $g_i(x)$, which can further be defined as a composition of other functions. This can be conveniently represented as a network structure, with arrows depicting the dependencies between variables. A widely

used type of composition is the *nonlinear weighted sum*, where $f(x) = K\left(\sum_i w_i g_i(x)\right)$, where K (commonly referred to as the activation function) is some predefined function, such as the hyperbolic tangent. It will be convenient for the following to refer to a collection of functions g_i as simply a vector $g = (g_1, g_2, \ldots, g_n)$.



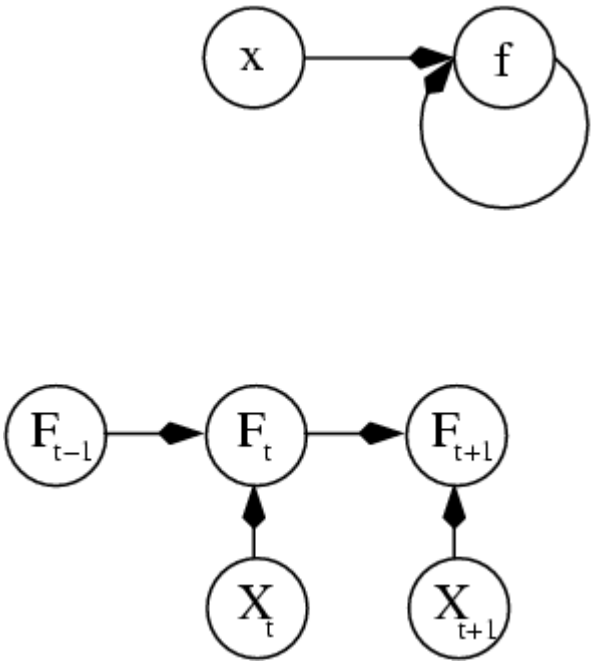
ANN dependency graph

This figure depicts such a decomposition of f, with dependencies between variables indicated by arrows. These can be interpreted in two ways.

The first view is the functional view: the input x is transformed into a 3-dimensional vector h, which is then transformed into a 2-dimensional vector g, which is finally transformed into f. This view is most commonly encountered in the context of optimization.

The second view is the probabilistic view: the random variable $F = f(G)$ depends upon the random variable $G = g(H)$, which depends upon $H = h(X)$, which depends upon the random variable X. This view is most commonly encountered in the context of graphical models.

The two views are largely equivalent. In either case, for this particular network architecture, the components of individual layers are independent of each other (e.g., the components of g are independent of each other given their input h). This naturally enables a degree of parallelism in the implementation.



Recurrent ANN dependency graph

Networks such as the previous one are commonly called feedforward, because their graph is a directed acyclic graph. Networks with cycles are commonly called recurrent. Such networks are commonly depicted in the manner shown at the top of the figure, where f is shown as being dependent upon itself. However, an implied temporal dependence is not shown.ANN depends on three basic critareas:

```
- Interconnection between different Layers of Neurons;
- Learning process of ANN;
- Activation Function;
```

Interconnection shows the relationship between single layer ,multiple layers of input output parameters of Neurons it shows the relationship of One to Many .it means same input can perform many outputs for different layer of architecture.

Learning

What has attracted the most interest in neural networks is the possibility of *learning*. Given a specific *task* to solve, and a *class* of functions, F, learning means using a set of *observations* to find $f^* \in F$ which solves the task in some *optimal* sense.

This entails defining a cost function $C : F \to \mathbb{R}$ such that, for the optimal solution f^*, $C(f^*) \leq C(f) \forall f \in F$ (i.e., no solution has a cost less than the cost of the optimal solution).

The cost function C is an important concept in learning, as it is a measure of how far away a particular solution is from an optimal solution to the problem to be solved. Learning algorithms search through the solution space to find a function that has the smallest possible cost.

For applications where the solution is dependent on some data, the cost must necessarily be a *function of the observations*, otherwise we would not be modelling anything related to the data. It is frequently defined as a statistic to which only approximations can be made. As a simple example, consider the problem of finding the model f, which minimizes $C=E\left[(f(x)-y)^2\right]$, for data pairs (x,y)drawn from some distribution $\mathcal{D}$. In practical situations we would only have Nsamples from $\mathcal{D}$and thus, for the above example, we would only minimize $\hat{C}=\frac{1}{N}\sum_{i=1}^{N}(f(x_i)-y_i)^2$. Thus, the cost is minimized over a sample of the data rather than the entire data set.

When $N\to\infty$some form of online machine learning must be used, where the cost is partially minimized as each new example is seen. While online machine learning is often used when $\mathcal{D}$is fixed, it is most useful in the case where the distribution changes slowly over time. In neural network methods, some form of online machine learning is frequently used for finite datasets.

Choosing a cost function

While it is possible to define some arbitrary, ad hoc cost function, frequently a particular cost will be used, either because it has desirable properties (such as convexity) or because it arises naturally from a particular formulation of the problem (e.g., in a probabilistic formulation the posterior probability of the model can be used as an inverse cost). Ultimately, the cost function will depend on the desired task. An overview of the three main categories of learning tasks is provided below.

Learning paradigms

There are three major learning paradigms, each corresponding to a particular abstract learning task. These are supervised learning, unsupervised learning and reinforcement learning.

Supervised learning

In supervised learning, we are given a set of example pairs $(x,y),x\in X,y\in Y$and the aim is to find a function $f:X\to Y$in the allowed class of functions that matches the examples. In other words, we wish to *infer* the mapping implied by the data; the cost function is related to the mismatch between our mapping and the data and it implicitly contains prior knowledge about the problem domain.

A commonly used cost is the mean-squared error, which tries to minimize the average squared error between the network's output, f(x), and the target value y over all the example pairs. When one tries to minimize this cost using gradient descent for the class of neural networks called multilayer perceptrons, one obtains the common and well-known backpropagation algorithm for training neural networks.

Tasks that fall within the paradigm of supervised learning are pattern recognition (also known as classification) and regression (also known as function approximation). The supervised learning paradigm is also applicable to sequential data (e.g., for speech and gesture recognition). This can be thought of as learning with a "teacher," in the form of a function that provides continuous feedback on the quality of solutions obtained thus far.

Basically supervised learning are classified in two types.These are error connection gradient descent and stochastic.

Error connection gradient descent are also classified into least mean square and backpropagation.

Unsupervised learning

In unsupervised learning, some data $x$is given and the cost function to be minimized, that can be any function of the data $x$and the network's output, f.

The cost function is dependent on the task (what we are trying to model) and our *a priori* assumptions (the implicit properties of our model, its parameters and the observed variables).

As a trivial example, consider the model $f(x)=a$, where $a$is a constant and the cost $C=E[(x-f(x))^2]$. Minimizing this cost will give us a value of $a$that is equal to the mean of the data. The cost function can be much more complicated. Its form depends on the application: for example, in compression it could be related to the mutual information between x and y, whereas in statistical modeling, it could be related to the posterior probability of the model given the data. (Note that in both of those examples those quantities would be maximized rather than minimized).

Tasks that fall within the paradigm of unsupervised learning are in general estimation problems; the applications include clustering, the estimation of statistical distributions, compression and filtering.

Reinforcement learning

In reinforcement learning, data $x$are usually not given, but generated by an agent's interactions with the environment. At each point in time t, the agent performs an action $y_t$and the environment generates an observation $x_t$and an instantaneous cost c_t, according to some (usually unknown) dynamics. The aim is to discover a *policy* for selecting actions that minimizes some measure of a long-term cost; i.e., the expected cumulative cost. The environment's dynamics and the long-term cost for each policy are usually unknown, but can be estimated.

More formally, the environment is modeled as a Markov decision process (MDP) with states $s_1,...,s_n \in S$and actions $a_1,...,a_m \in A$with the following probability distributions: the instantaneous cost distribution $P(c_t|s_t)$, the observation distribution $P(x_t|s_t)$and the

transition $P(s_{t+1}|s_t,a_t)$, while a policy is defined as conditional distribution over actions given the observations. Taken together, the two define a Markov chain (MC). The aim is to discover the policy that minimizes the cost; i.e., the MC for which the cost is minimal.

ANNs are frequently used in reinforcement learning as part of the overall algorithm.

Tasks that fall within the paradigm of reinforcement learning are control problems, games and other sequential decision making tasks.

Learning algorithms

Training a neural network model essentially means selecting one model from the set of allowed models (or, in a Bayesian framework, determining a distribution over the set of allowed models) that minimizes the cost criterion. There are numerous algorithms available for training neural network models; most of them can be viewed as a straightforward application of optimization theory and statistical estimation. Recent developments in this field use particle swarm optimization and other swarm intelligence techniques.

Most of the algorithms used in training artificial neural networks employ some form of gradient descent. This is done by simply taking the derivative of the cost function with respect to the network parameters and then changing those parameters in a gradient-related direction.

Evolutionary methods, simulated annealing, expectation-maximization and non-parametric methods are some commonly used methods for training neural networks.

Temporal perceptual learning relies on finding temporal relationships in sensory signal streams. In an environment, statistically salient temporal correlations can be found by monitoring the arrival times of sensory signals. This is done by the perceptual network.

Employing artificial neural networks

Perhaps the greatest advantage of ANNs is their ability to be used as an arbitrary function approximation mechanism that 'learns' from observed data. However, using them is not so straightforward and a relatively good understanding of the underlying theory is essential.

- Choice of model: This will depend on the data representation and the application. Overly complex models tend to lead to problems with learning.
- Learning algorithm: There are numerous trade-offs between learning algorithms. Almost any algorithm will work well with the *correct hyperparameters* for training on a particular fixed data set. However selecting and tuning an algorithm for training on unseen data requires a significant amount of experimentation.
- Robustness: If the model, cost function and learning algorithm are selected appropriately the resulting ANN can be extremely robust.

With the correct implementation, ANNs can be used naturally in online learning and large data set applications. Their simple implementation and the existence of mostly local dependencies exhibited in the structure allows for fast, parallel implementations in hardware.

Applications

The utility of artificial neural network models lies in the fact that they can be used to infer a function from observations. This is particularly useful in applications where the complexity of the data or task makes the design of such a function by hand impractical.

Real-life applications

The tasks artificial neural networks are applied to tend to fall within the following broad categories:

- Function approximation, or regression analysis, including time series prediction, fitness approximation and modeling.
- Classification, including pattern and sequence recognition, novelty detection and sequential decision making.
- Data processing, including filtering, clustering, blind source separation and compression.
- Robotics, including directing manipulators, Computer numerical control.

Application areas include system identification and control (vehicle control, process control), quantum chemistry, game-playing and decision making (backgammon, chess, racing), pattern recognition (radar systems, face identification, object recognition and more), sequence recognition (gesture, speech, handwritten text recognition), medical diagnosis, financial applications (automated trading systems), data mining (or knowledge discovery in databases, "KDD"), visualization and e-mail spam filtering.

Neural networks and neuroscience

Theoretical and computational neuroscience is the field concerned with the theoretical analysis and computational modeling of biological neural systems. Since neural systems are intimately related to cognitive processes and behavior, the field is closely related to cognitive and behavioral modeling.

The aim of the field is to create models of biological neural systems in order to understand how biological systems work. To gain this understanding, neuroscientists strive to make a link between observed biological processes (data), biologically plausible mechanisms for neural processing and learning (biological neural network models) and theory (statistical learning theory and information theory).

Types of models

Many models are used in the field defined at different levels of abstraction and modeling different aspects of neural systems. They range from models of the short-term behavior of individual neurons, models of how the dynamics of neural circuitry arise from interactions between individual neurons and finally to models of how behavior can arise from abstract neural modules that represent complete subsystems. These include models of the long-term, and short-term plasticity, of neural systems and their relations to learning and memory from the individual neuron to the system level.

Current research

While initial research had been concerned mostly with the electrical characteristics of neurons, a particularly important part of the investigation in recent years has been the exploration of the role of neuromodulators such as dopamine, acetylcholine, and serotonin on behavior and learning.

Biophysical models, such as BCM theory, have been important in understanding mechanisms for synaptic plasticity, and have had applications in both computer science and neuroscience. Research is ongoing in understanding the computational algorithms used in the brain, with some recent biological evidence for radial basis networks and neural backpropagation as mechanisms for processing data.

Computational devices have been created in CMOS for both biophysical simulation and neuromorphic computing. More recent efforts show promise for creating nanodevices for very large scale principal components analyses and convolution. If successful, these effort could usher in a new era of neural computing that is a step beyond digital computing, because it depends on learning rather than programming and because it is fundamentally analog rather than digital even though the first instantiations may in fact be with CMOS digital devices.

Neural network software

Neural network software is used to simulate, research, develop and apply artificial neural networks, biological neural networks and in some cases a wider array of adaptive systems.

Types of artificial neural networks

Artificial neural network types vary from those with only one or two layers of single direction logic, to complicated multi–input many directional feedback loop and layers. On the whole, these systems use algorithms in their programming to determine control and organization of their functions. Some may be as simple, one neuron layer with an input and an output, and others can mimic complex systems such as dANN, which can

mimic chromosomal DNA through sizes at cellular level, into artificial organisms and simulate reproduction, mutation and population sizes.

Most systems use "weights" to change the parameters of the throughput and the varying connections to the neurons. Artificial neural networks can be autonomous and learn by input from outside "teachers" or even self-teaching from written in rules.

Theoretical properties

Computational power

The multi-layer perceptron (MLP) is a universal function approximator, as proven by the Cybenko theorem. However, the proof is not constructive regarding the number of neurons required or the settings of the weights.

Work by Hava Siegelmann and Eduardo D. Sontag has provided a proof that a specific recurrent architecture with rational valued weights (as opposed to full precision real number-valued weights) has the full power of a Universal Turing Machine using a finite number of neurons and standard linear connections. They have further shown that the use of irrational values for weights results in a machine with super-Turing power.

Capacity

Artificial neural network models have a property called 'capacity', which roughly corresponds to their ability to model any given function. It is related to the amount of information that can be stored in the network and to the notion of complexity.

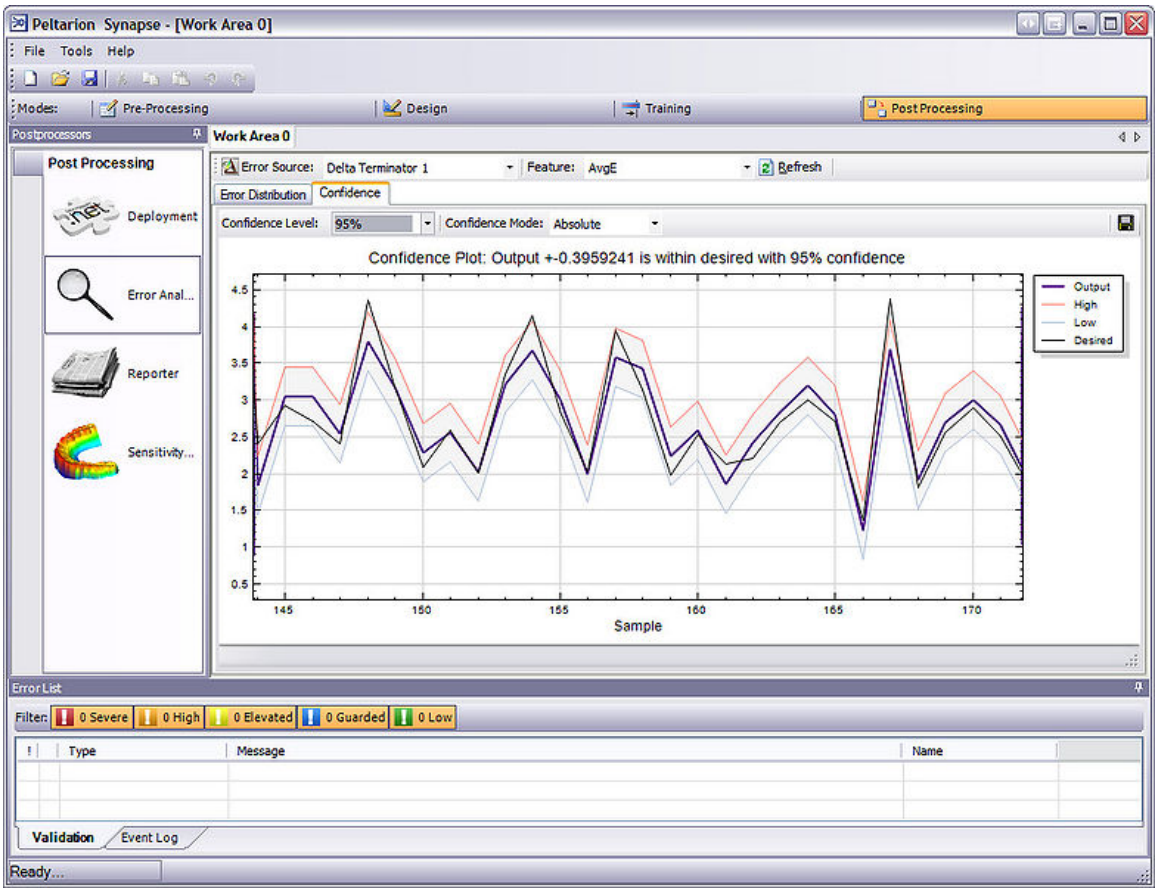
Convergence

Nothing can be said in general about convergence since it depends on a number of factors. Firstly, there may exist many local minima. This depends on the cost function and the model. Secondly, the optimization method used might not be guaranteed to converge when far away from a local minimum. Thirdly, for a very large amount of data or parameters, some methods become impractical. In general, it has been found that theoretical guarantees regarding convergence are an unreliable guide to practical application.

Generalization and statistics

In applications where the goal is to create a system that generalizes well in unseen examples, the problem of over-training has emerged. This arises in convoluted or over-specified systems when the capacity of the network significantly exceeds the needed free parameters. There are two schools of thought for avoiding this problem: The first is to use cross-validation and similar techniques to check for the presence of overtraining and optimally select hyperparameters such as to minimize the generalization error. The

second is to use some form of *regularization*. This is a concept that emerges naturally in a probabilistic (Bayesian) framework, where the regularization can be performed by selecting a larger prior probability over simpler models; but also in statistical learning theory, where the goal is to minimize over two quantities: the 'empirical risk' and the 'structural risk', which roughly corresponds to the error over the training set and the predicted error in unseen data due to overfitting.



Confidence analysis of a neural network

Supervised neural networks that use an MSE cost function can use formal statistical methods to determine the confidence of the trained model. The MSE on a validation set can be used as an estimate for variance. This value can then be used to calculate the confidence interval of the output of the network, assuming a normal distribution. A confidence analysis made this way is statistically valid as long as the output probability distribution stays the same and the network is not modified.

By assigning a softmax activation function on the output layer of the neural network (or a softmax component in a component-based neural network) for categorical target variables, the outputs can be interpreted as posterior probabilities. This is very useful in classification as it gives a certainty measure on classifications.

The softmax activation function is:

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{c} e^{x_j}}$$

Dynamic properties

Various techniques originally developed for studying disordered magnetic systems (i.e., the spin glass) have been successfully applied to simple neural network architectures, such as the Hopfield network. Influential work by E. Gardner and B. Derrida has revealed many interesting properties about perceptrons with real-valued synaptic weights, while later work by W. Krauth and M. Mezard has extended these principles to binary-valued synapses.

Criticism

A common criticism of artificial neural networks, particularly in robotics, is that they require a large diversity of training for real-world operation. Dean Pomerleau, in his research presented in the paper "Knowledge-based Training of Artificial Neural Networks for Autonomous Robot Driving," uses a neural network to train a robotic vehicle to drive on multiple types of roads (single lane, multi-lane, dirt, etc.). A large amount of his research is devoted to (1) extrapolating multiple training scenarios from a single training experience, and (2) preserving past training diversity so that the system does not become overtrained (if, for example, it is presented with a series of right turns – it should not learn to always turn right). These issues are common in neural networks that must decide from amongst a wide variety of responses.

A. K. Dewdney, a former *Scientific American* columnist, wrote in 1997, "Although neural nets do solve a few toy problems, their powers of computation are so limited that I am surprised anyone takes them seriously as a general problem-solving tool." (Dewdney, p. 82)

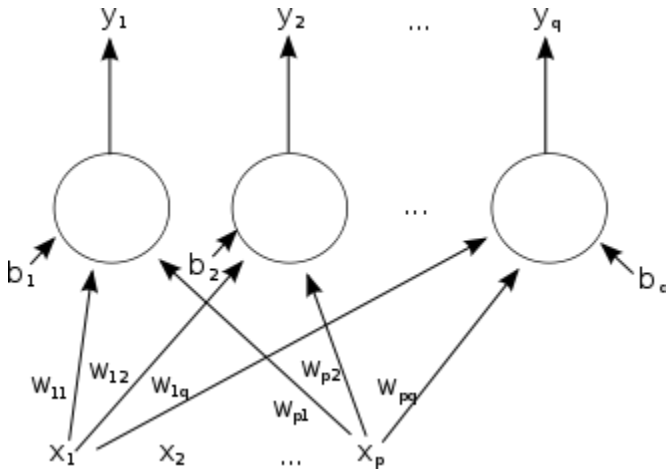
Arguments for Dewdney's position are that to implement large and effective software neural networks, much processing and storage resources need to be committed. While the brain has hardware tailored to the task of processing signals through a graph of neurons, simulating even a most simplified form on Von Neumann technology may compel a NN designer to fill many millions of database rows for its connections - which can lead to abusive RAM and HD necessities. Furthermore, the designer of NN systems will often need to simulate the transmission of signals through many of these connections and their associated neurons - which must often be matched with incredible amounts of CPU processing power and time. While neural networks often yield *effective* programs, they too often do so at the cost of time and money *efficiency*.

Arguments against Dewdney's position are that neural nets have been successfully used to solve many complex and diverse tasks, ranging from autonomously flying aircraft to detecting credit card fraud. Technology writer Roger Bridgman commented on Dewdney's statements about neural nets:

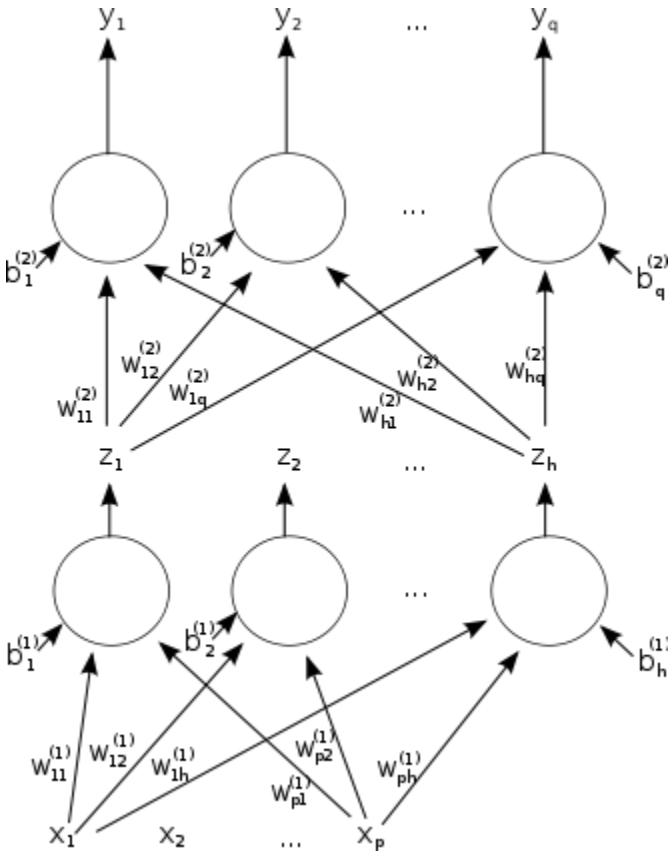
Neural networks, for instance, are in the dock not only because they have been hyped to high heaven, (what hasn't?) but also because you could create a successful net without understanding how it worked: the bunch of numbers that captures its behaviour would in all probability be "an opaque, unreadable table...valueless as a scientific resource". In spite of his emphatic declaration that science is not technology, Dewdney seems here to pillory neural nets as bad science when most of those devising them are just trying to be good engineers. An unreadable table that a useful machine could read would still be well worth having.

Some other criticisms came from believers of hybrid models (combining neural networks and symbolic approaches). They advocate the intermix of these two approaches and believe that hybrid models can better capture the mechanisms of the human mind (Sun and Bookman 1994).

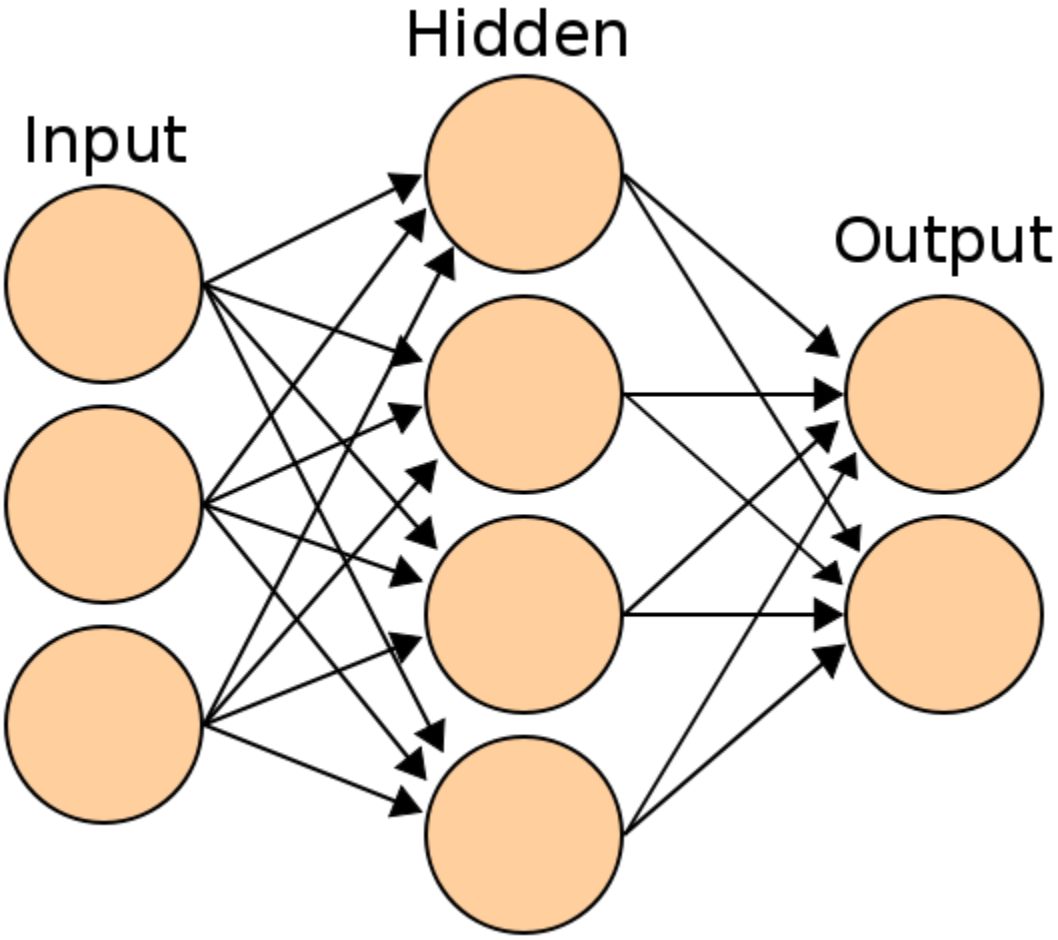
Gallery

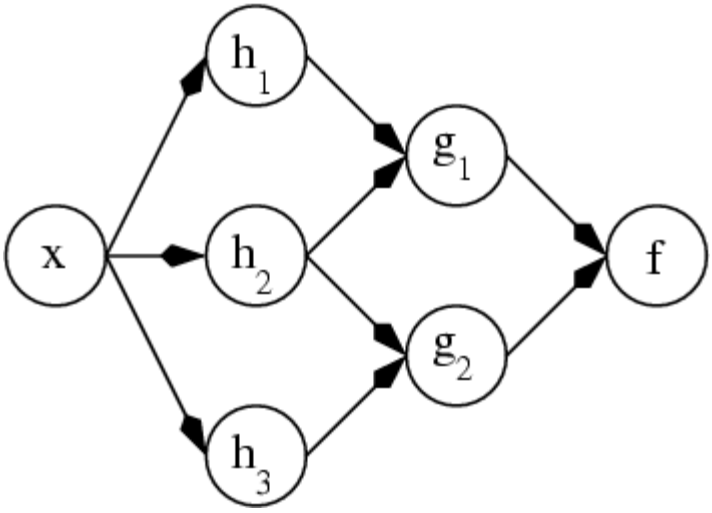


A single-layer feedforward artificial neural network. Arrows originating from $x_2$are omitted for clarity. There are p inputs to this network and q outputs. There is no activation function (or equivalently, the activation function is $g(x)=x$). In this system, the value of the qth output, $y_q$would be calculated as $y_q=\sum(x_i*w_{iq})$



A two-layer feedforward artificial neural network.


Hidden
Input
Output


h1
g1
x
h2
f
g2
h3