

Ansys Twin Builder basicセミナー

制御シミュレーション編

サイバネットシステム株式会社



- ・本セミナーの目的

本セミナーはAnsys Twin Builderの基本操作を習得頂いた方がシミュレーションによる制御系設計を実施していただけるようになる為のきっかけとしていただくことを目的としております。

- ・本セミナーで使用するライセンス

Ansys Twin Builder Pro



第1章 はじめに

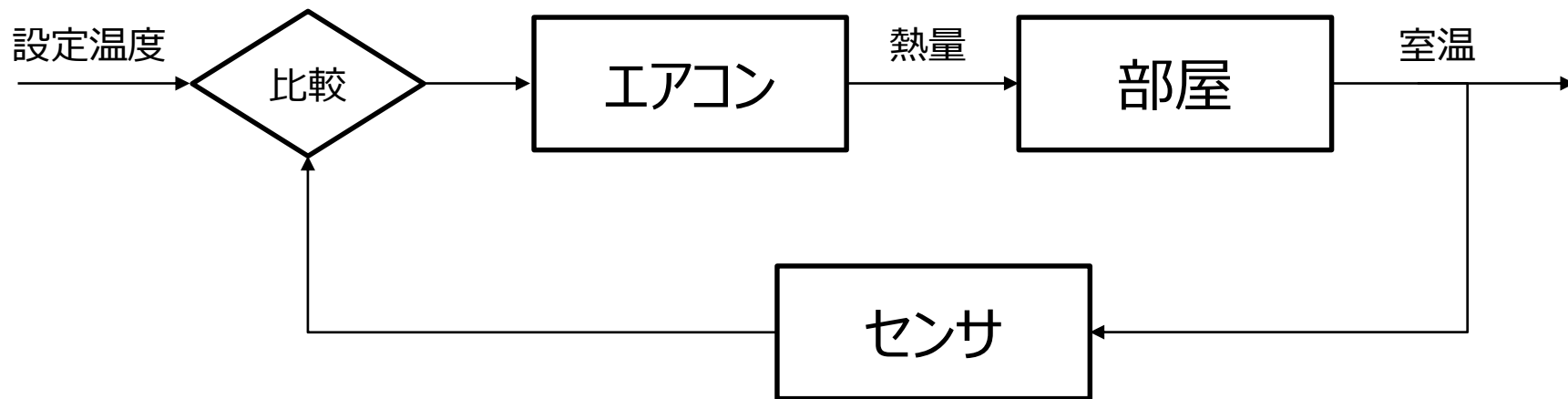
自動制御の例



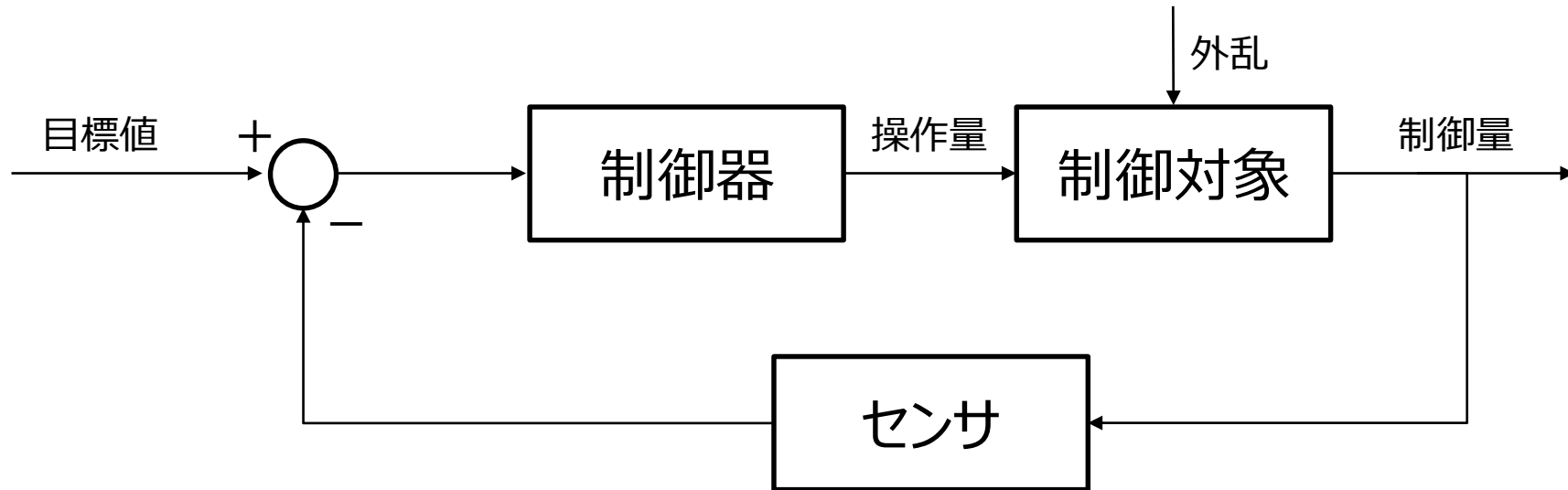
シーケンス制御



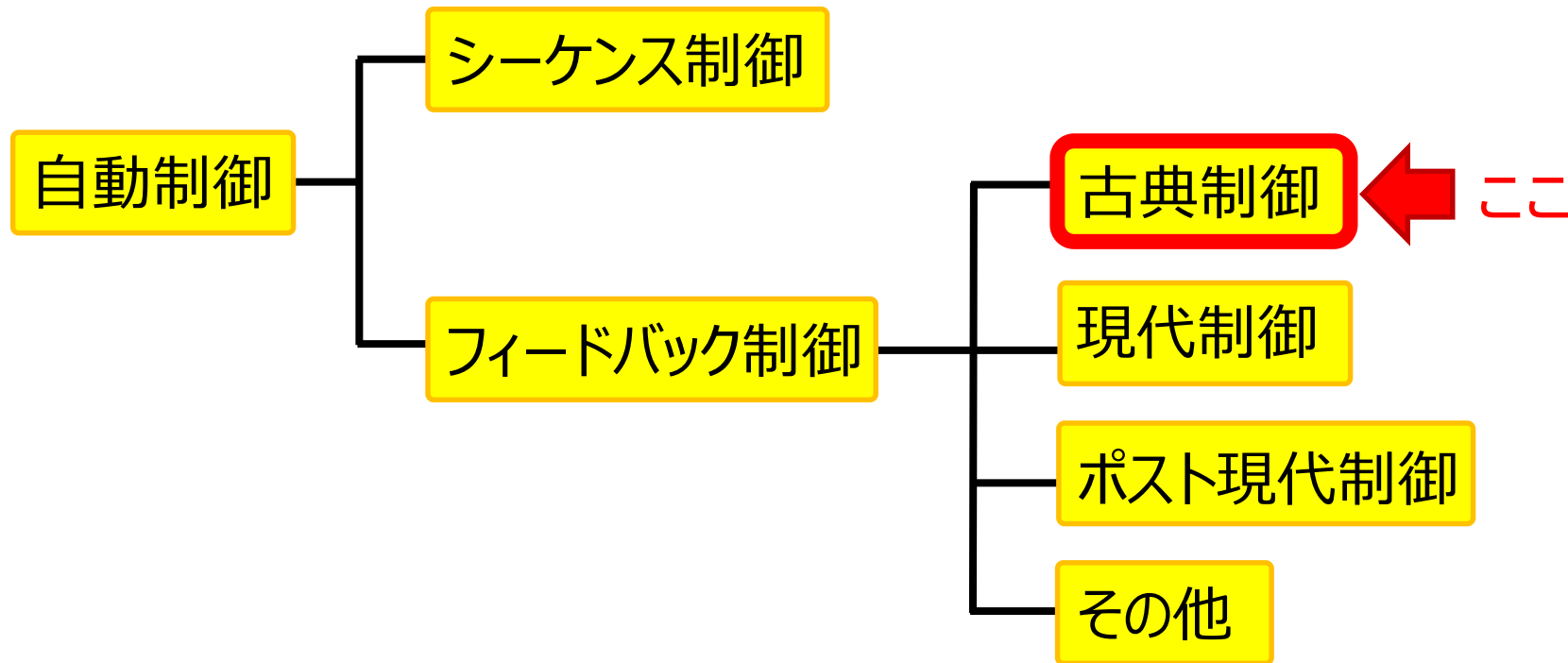
フィードバック制御



フィードバック制御



制御の分類



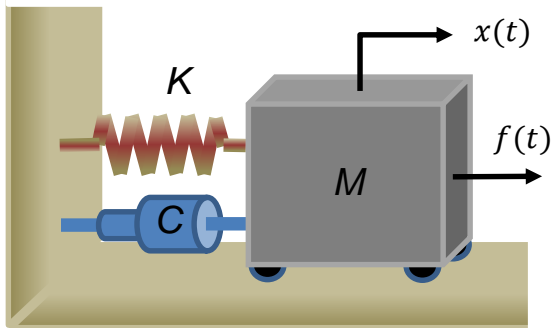


第2章 動的システムとラプラス変換

動的システム

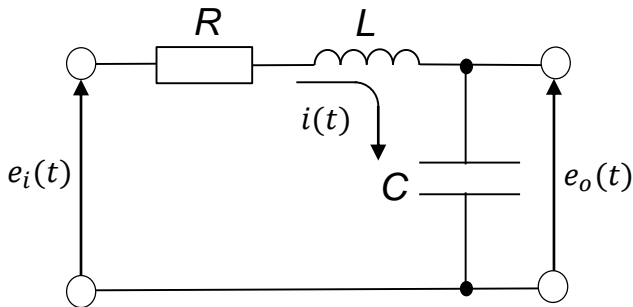
- 現在の入力だけでなく過去の状態にも依存する
 - 微分方程式で表される現象

質量-ばね-ダンパ系



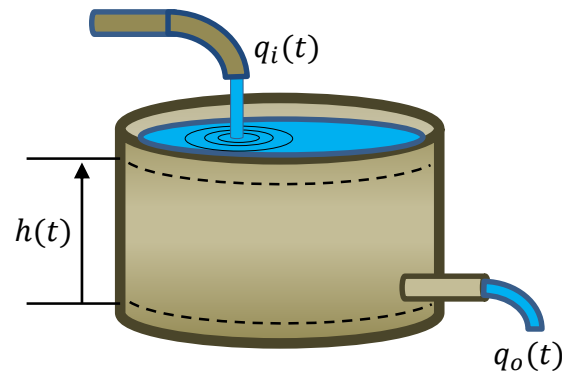
$$M \frac{d^2 x(t)}{dt^2} + C \frac{dx(t)}{dt} + Kx(t) = f(t)$$

RLC回路系



$$LC \frac{d^2 e_o(t)}{dt^2} + RC \frac{de_o(t)}{dt} + e_o(t) = e_i(t)$$

水位系



$$A \frac{dh(t)}{dt} = q_i(t) - q_o(t)$$

動的システムと微分方程式

- 前述の動的システムに現れた微分方程式を一般化すると

$$a_n \frac{d^n y(t)}{dt^n} + a_{n-1} \frac{d^{n-1} y(t)}{dt^{n-1}} + \cdots + a_1 \frac{dy(t)}{dt} + a_0 y(t)$$

出力 $y(t)$

$$= b_m \frac{d^m u(t)}{dt^m} + b_{m-1} \frac{d^{m-1} u(t)}{dt^{m-1}} + \cdots + b_1 \frac{du(t)}{dt} + b_0 u(t)$$

入力 $u(t)$

- 高階の線形常微分方程式となっているので
一般に解を求めることが難しい. . .

ラプラス変換

- 以下のように関数 $f(t)$ に対して積分を実施します. s は複素数

$$F(s) = \int_0^{\infty} f(t) e^{-st} dt \quad F(s) = \mathcal{L}[f(t)] \quad \text{便宜上このように表現します}$$

- ラプラス変換を使用すると関数に対する微分積分を乗算と除算に変換することができます.

$$\begin{aligned} \mathcal{L}\left[\frac{df(t)}{dt}\right] &= \int_0^{\infty} (f(t))' e^{-st} dt = [f(t)e^{-st}]_0^{\infty} - \int_0^{\infty} f(t)(e^{-st})' dt \\ &= [f(t)e^{-st}]_0^{\infty} - \int_0^{\infty} f(t)(e^{-st})' dt = \boxed{-f(0)} + s \int_0^{\infty} f(t)e^{-st} dt \end{aligned}$$

$\mathcal{L}[f(t)]$

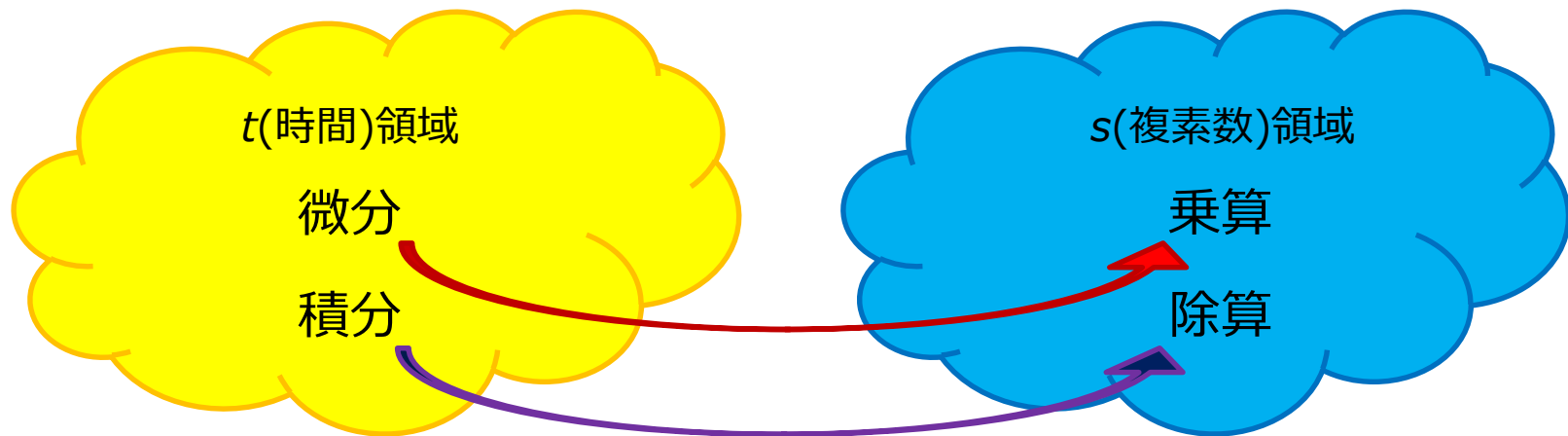
ラプラス変換

- ラプラス変換により

微分 $\mathcal{L}\left[\frac{df(t)}{dt}\right] = sF(s)$

積分 $\mathcal{L}\left[\int_0^t g(\tau)d\tau\right] = \frac{1}{s}G(s)$

※積分は微分の逆演算なので簡単な計算で除算になることを示せます。



ラプラス変換

- 前述の微分方程式をラプラス変換すると

$$a_n \frac{d^n y(t)}{dt^n} + a_{n-1} \frac{d^{n-1} y(t)}{dt^{n-1}} + \cdots + a_1 \frac{dy(t)}{dt} + a_0 y(t)$$

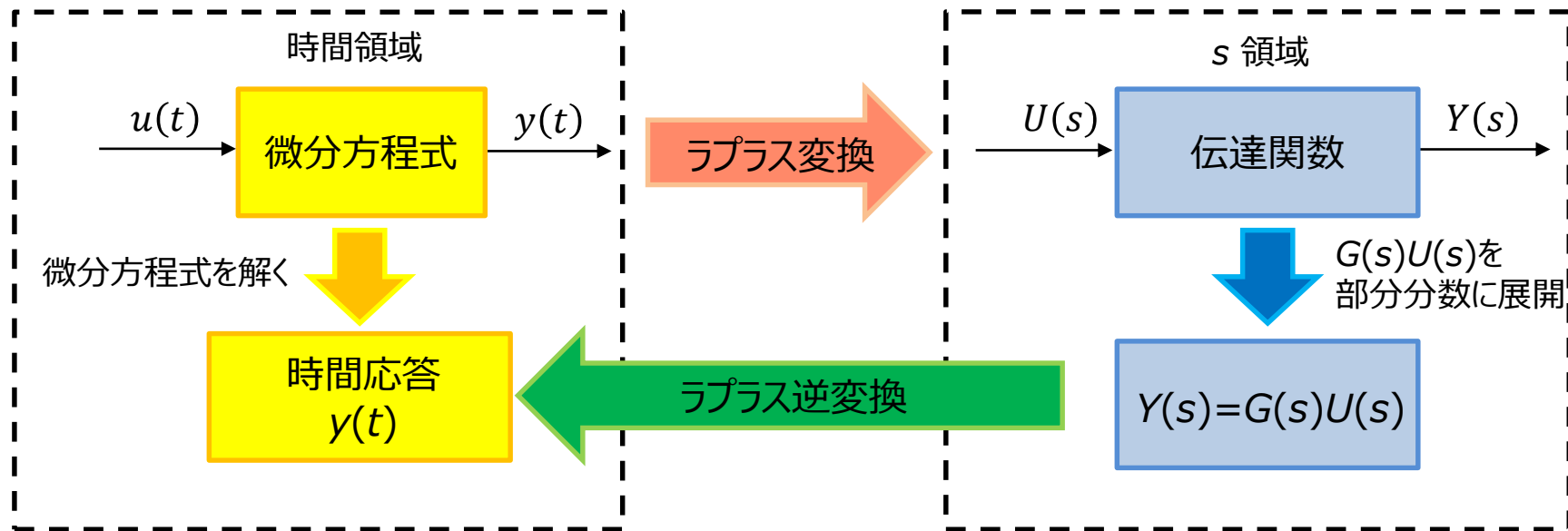
$$= b_m \frac{d^m u(t)}{dt^m} + b_{m-1} \frac{d^{m-1} u(t)}{dt^{m-1}} + \cdots + b_1 \frac{du(t)}{dt} + b_0 u(t)$$

$$(a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0) Y(s) = (b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1 s + b_0) U(s)$$

$$Y(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0} U(s)$$

伝達関数 $G(s)$

ラプラス変換



- ラプラス変換を使うと部分分数分解などの簡単な代数操作で時間応答を求めることができます。

ラプラス変換

- 代表的な関数とラプラス変換表

原関数 : $f(t)$	像関数 : $\mathcal{L}[f(t)]$	原関数 : $f(t)$	像関数 : $\mathcal{L}[f(t)]$
単位インパルス 信号 : $\delta(t)$	1	$\cos \omega t$	$\frac{s}{s^2 + \omega^2}$
単位ステップ 信号 : $u(t)$	$\frac{1}{s}$	te^{-at}	$\frac{1}{(s + a)^2}$
t	$\frac{1}{s^2}$	$\frac{t^n}{n!}$	$\frac{1}{s^{n+1}}$
e^{-at}	$\frac{1}{s + a}$	$e^{-at} \sin \omega t$	$\frac{\omega}{(s + a)^2 + \omega^2}$
$\sin \omega t$	$\frac{\omega}{s^2 + \omega^2}$	$e^{-at} \cos \omega t$	$\frac{s + a}{(s + a)^2 + \omega^2}$

ラプラス変換

- 簡単な例で確認

次のような(1次遅れの)伝達関数に単位ステップを加えた時の応答を求めてみます。

$$G(s) = \frac{2}{s+4}$$

単位ステップなので $u(t) = 1$ になります。ラプラス変換すると $U(s) = 1/s$ なので出力は

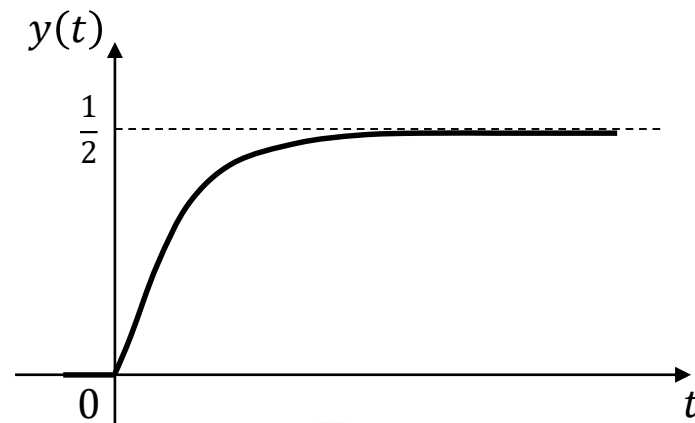
$$Y(s) = G(s)U(s) = \frac{2}{s+4} \cdot \frac{1}{s}$$

部分分数に分解すると

$$Y(s) = G(s)U(s) = \frac{2}{s+4} \cdot \frac{1}{s} = \frac{1}{2} \cdot \frac{1}{s} - \frac{1}{2} \cdot \frac{1}{s+4}$$

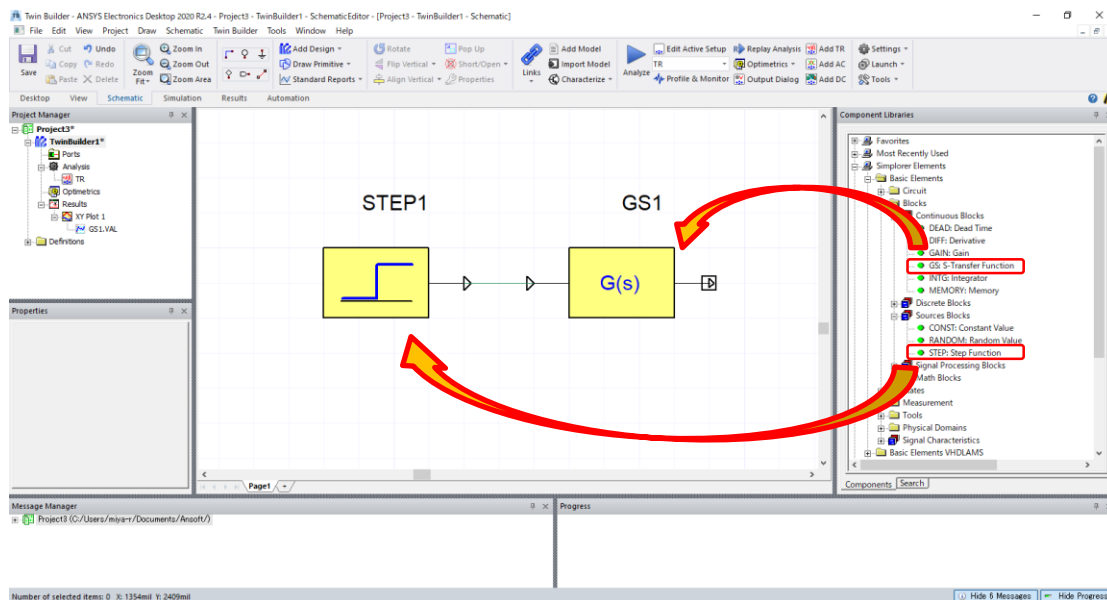
時間領域に戻しますと

$$y(t) = \frac{1}{2} \cdot 1 - \frac{1}{2} \cdot e^{-4t} = \frac{1}{2}(1 - e^{-4t})$$



ラプラス変換

- シミュレーションで確認
 - STEPブロックとGSブロックを配置して結線します。



ラプラス変換

- シミュレーションで確認
 - STEPブロックとG(s)ブロックの設定を以下のように指定します。

Step Time: 0 s
Final Time: 1
Init Time: 0

Parameters - STEP1 - Step Function

Parameters | AC - Parameters | Output / Display

Name: STEP1 ☒ Show Name

Parameters

Step Time	0	s
Final Value	1	
Init Value	0	

Sample Time ☒ Use System Sample Time 0 s ☐ Use Pin

Outputs ☒ Block Output Signal

OK キャンセル

Order: 0
B[0] : 2

Order: 1
A[0] : 4
A[1] : 1

Parameters - GS1 - S-Transfer Function

Parameters | Output / Display

Name: GS1 ☒ Show Name

Input Signal: STEP1.VAL ☒ Use Pin

Numerator

Order	0
Coefficient	Value
B[0]	2

Denominator

Order	1
Coefficient	Value
A[0]	4
A[1]	1

Sample Time ☒ Use System Sample Time 0 s ☐ Use Pin

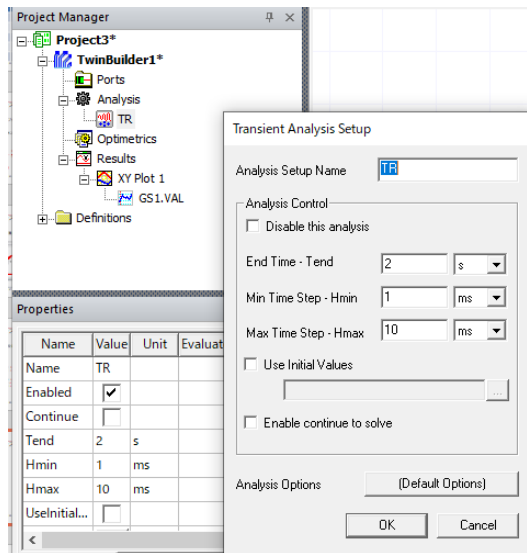
Outputs ☒ Block Output Signal

OK キャンセル

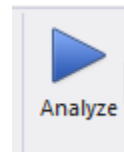
$$G(s) = \frac{2}{s + 4}$$

ラプラス変換

- シミュレーションで確認
 - Transient Analysis Setupを以下のように設定してシミュレーションを実行します。(Analyzeボタンをクリックします。)



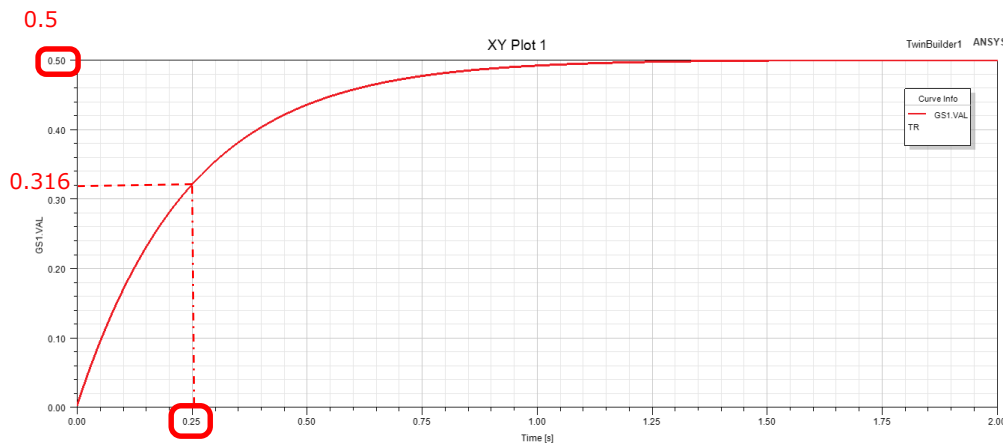
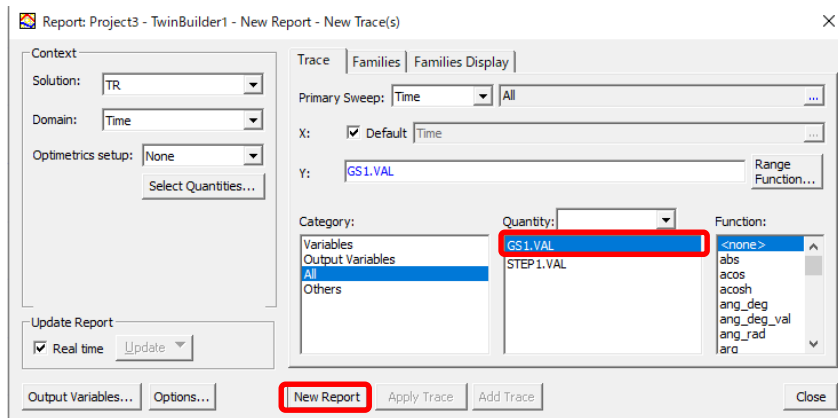
End Time - Tend: 2 s
Min Time Step - Hmin: 1 ms
Max Time Step - Hmax: 10ms



クリックします

ラプラス変換

- シミュレーションで確認
 - [Result]>[Create Standard Report]>[Rectangular Plot]をクリックします。



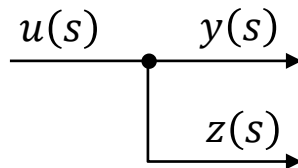
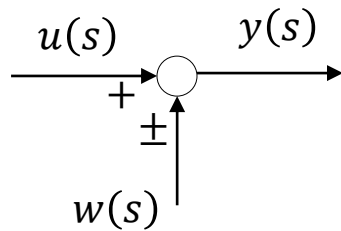
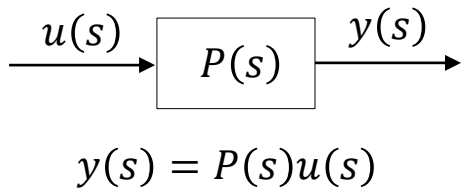
$$y(t) = \frac{1}{2}(1 - e^{-4t}) = \frac{1}{2}(1 - e^{-\frac{1}{0.25}t})$$



第3章 フィードバック制御の安定性

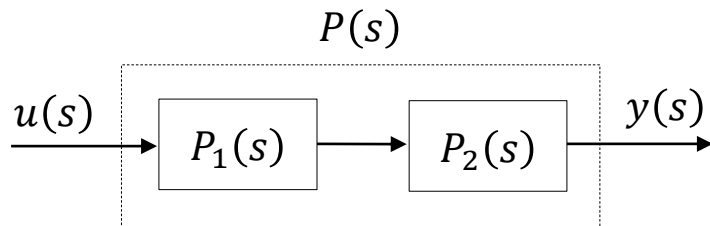
ブロック線図

- ブロック線図の結合について
 - 制御系の設計やシミュレーションでよくでるブロック線図の性質についてここで簡単に確認しておきます。

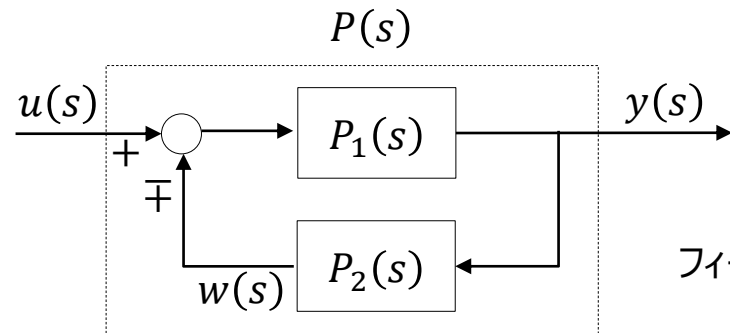


ブロック線図

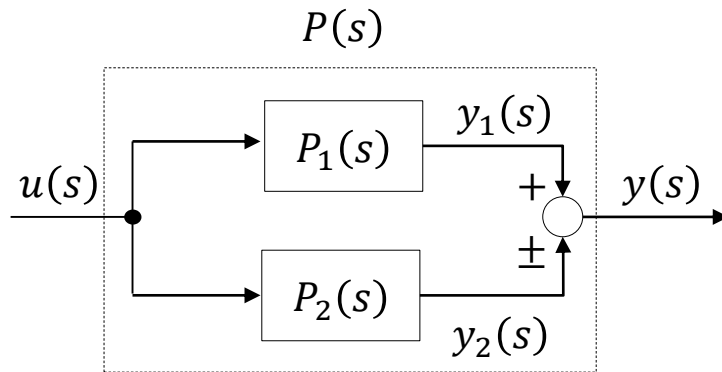
- 伝達関数の結合について



直列結合 $P(s) = P_1(s)P_2(s)$



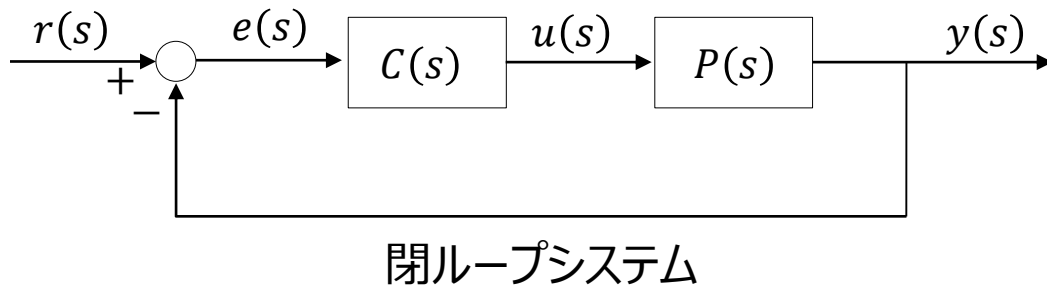
フィードバック結合 $P(s) = \frac{P_1(s)}{1 \pm P_1(s)P_2(s)}$



並列結合 $P(s) = P_1(s) \pm P_2(s)$

フィードバック制御系

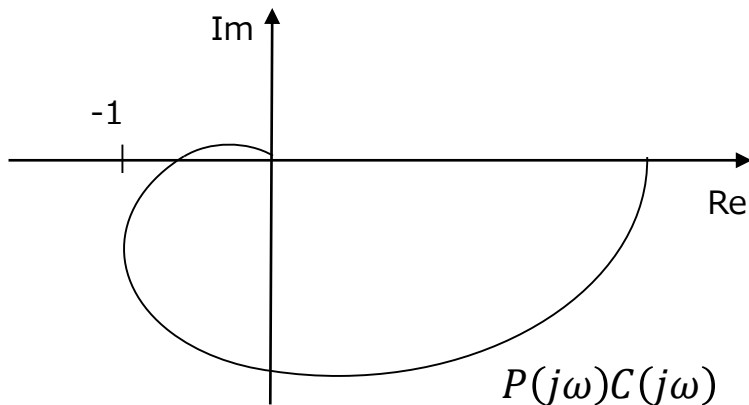
- 周波数領域における安定性
 - ここでフィードバック制御系の安定性について触れます。



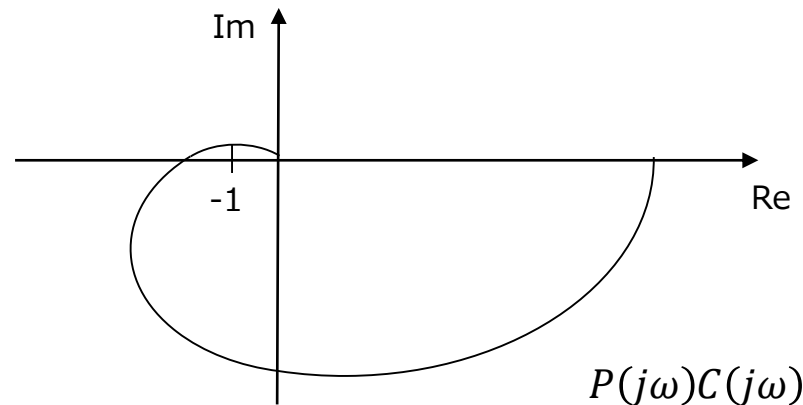
フィードバック結合
$$y(s) = \frac{P(s)C(s)}{1 + P(s)C(s)} r(s)$$

フィードバック制御系

- (簡略化された)ナイキストの安定判別法
 - 前述の閉ループシステムが安定であるためには, $P(s)C(s)$ が安定であり, $P(s)C(s)$ のベクトル軌跡が点 $(-1, 0)$ を常に左側に見ることである.



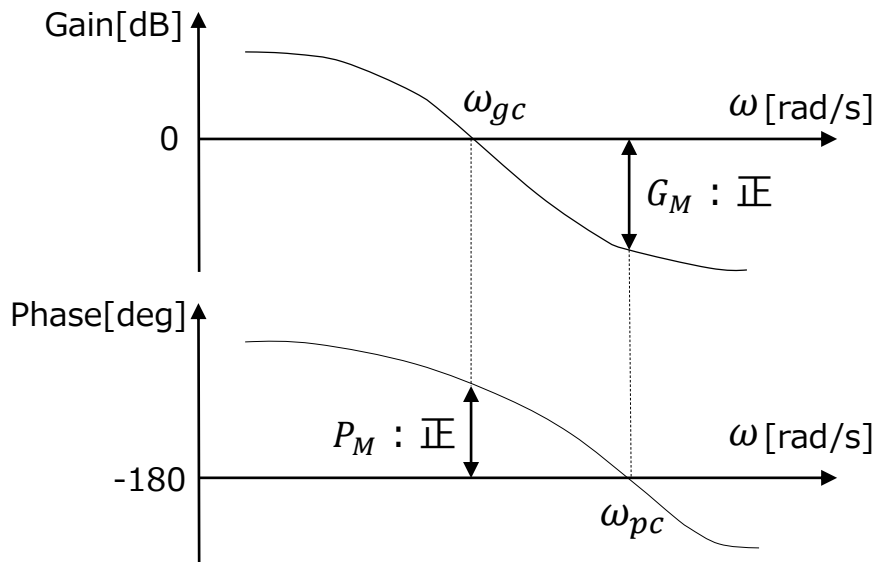
閉ループシステムが安定



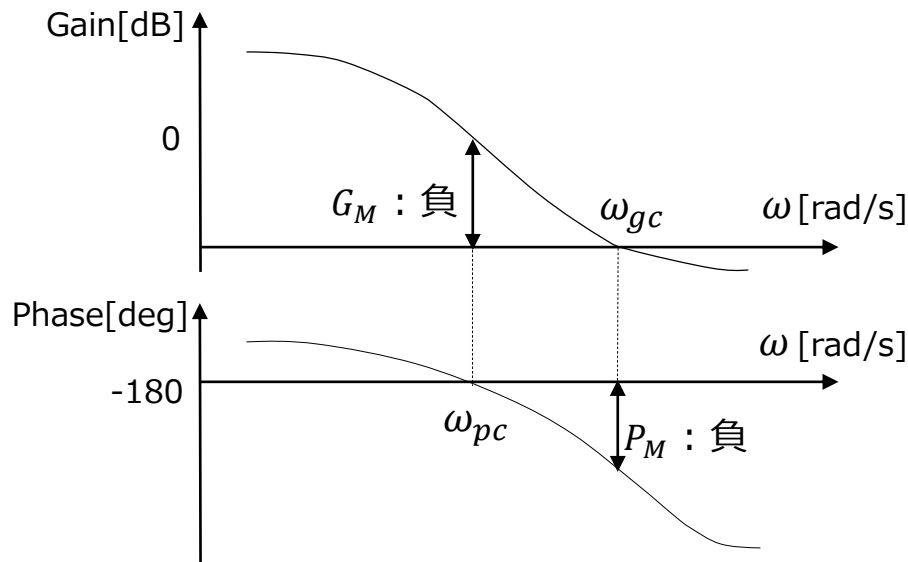
閉ループシステムが不安定

安定余裕

- 安定余裕については前述のナイキストの安定判別法のベクトル軌跡でも確認することができますが、ここではボード線図による見方を説明します。



閉ループシステムが安定な場合



閉ループシステムが不安定な場合

安定余裕

- 各用語について

ω_{gc} : ゲイン交差周波数

G_M : ゲイン余裕

ω_{pc} : 位相交差周波数

P_M : 位相余裕

シミュレーションによる確認

- 例

制御対象の伝達関数

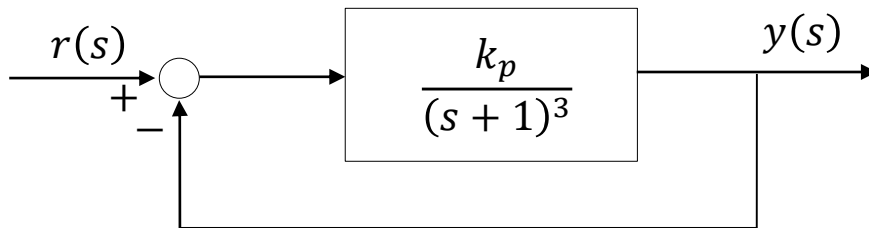
$$P(s) = \frac{1}{(s+1)^3}$$

コントローラの伝達関数を

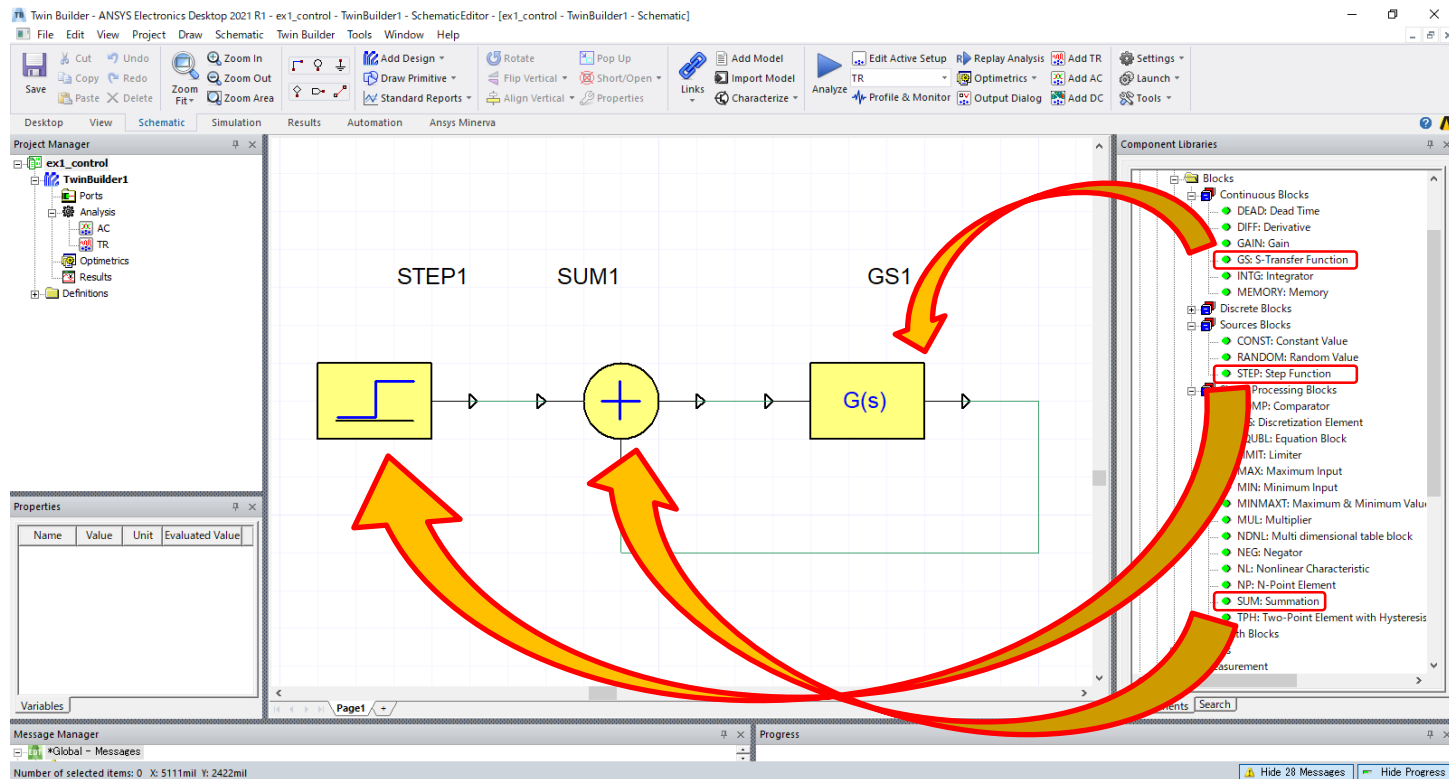
$$C(s) = k_p, \quad k_p > 0$$

とします.

このような閉ループ系のモデルを
TwinBuilderでモデル化します.



シミュレーションによる確認



シミュレーションによる確認

- 各ブロックの設定を以下のように指定します。

STEPブロック

Parameters - STEP1 - Step Function

Name: STEP1 ☒ Show Name

Parameters

Step Time: 0 s

Final Value: 1

Init Value: 0

Sample Time: ☒ Use System Sample Time, 0 s, ☐ Use Pin

Outputs: ☒ Block Output Signal

OK キャンセル

Step Time: 0 s
Final Time: 1
Init Time: 0

SUMブロック

Parameters - SUM1 - Summation

Name: SUM1 ☒ Show Name

Parameters

Name	Use Pin	Sign	Input Signal
INPUT[0]	<input checked="" type="checkbox"/>	+	STEP1.VAL
INPUT[1]	<input checked="" type="checkbox"/>	-	GS1.VAL
INPUT[2]	<input type="checkbox"/>		_Empty
INPUT[3]	<input type="checkbox"/>	+	_Empty
INPUT[4]	<input type="checkbox"/>	+	_Empty
INPUT[5]	<input type="checkbox"/>	+	_Empty
INPUT[6]	<input type="checkbox"/>	+	_Empty
INPUT[7]	<input type="checkbox"/>	+	_Empty

Sample Time: ☒ Use System Sample Time, 0 s, ☐ Use Pin

Outputs: ☒ Block Output Signal

OK キャンセル

Input[0] ☒ +
Input[1] ☒ -

GSブロック

Parameters - GS1 - S-Transfer Function

Name: GS1 ☒ Show Name

Input Signal: SUM1.VAL ☒ Use Pin

Numerator

Order	Coefficient	Value
0	B[0]	1

Denominator

Order	Coefficient	Value
0	A[0]	1
1	A[1]	3
2	A[2]	3
3	A[3]	1

Sample Time: ☒ Use System Sample Time, 0 s, ☐ Use Pin

Outputs: ☒ Block Output Signal

OK キャンセル

Order: 0
B[0] : 1

Order: 3
A[0] : 1
A[1] : 3
A[2] : 3
A[3] : 1

シミュレーションによる確認

- TR解析とAC解析の設定をします。

Transient Analysis Setup

Analysis Setup Name

Analysis Control

☐ Disable this analysis

End Time - Tend

Min Time Step - Hmin

Max Time Step - Hmax

☐ Use Initial Values

☐ Enable continue to solve

Analysis Options

AC Analysis Setup

Analysis Setup Name

Analysis Control

☐ Disable this analysis

Start frequency - FStart

Stop frequency - FEnd

Frequency step - Fstep

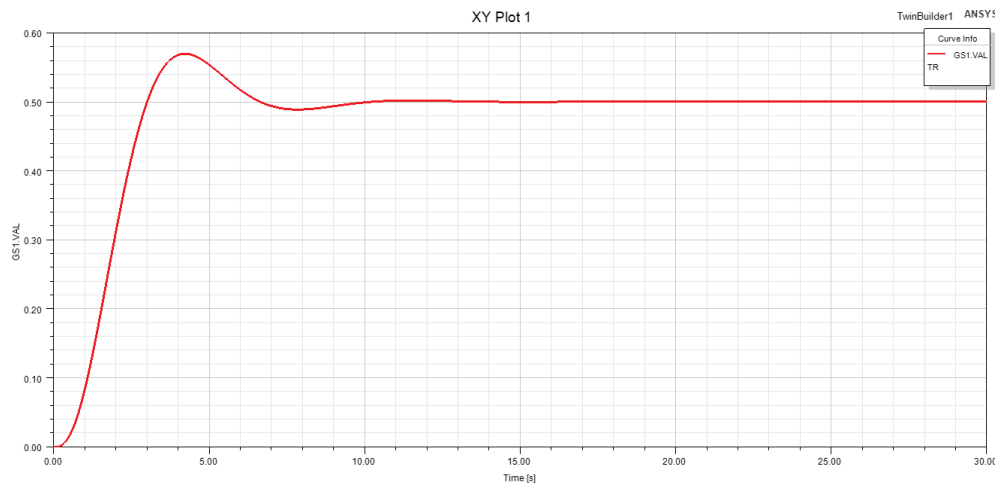
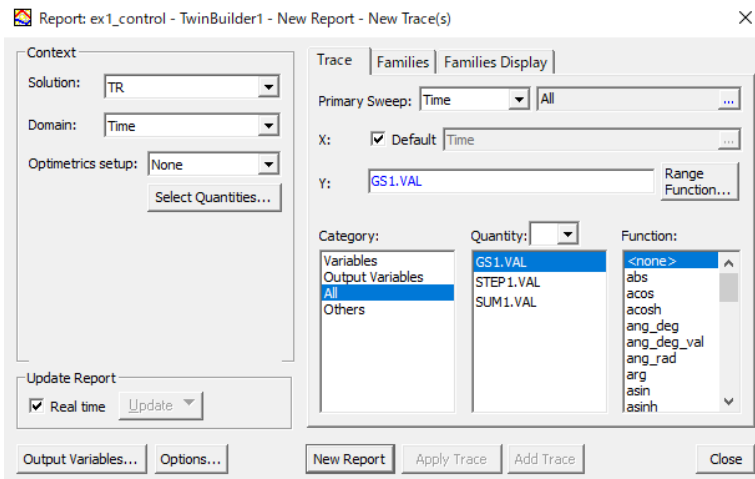
AC sweep type

☐ Enable continue to solve

Analysis Options

シミュレーションによる確認

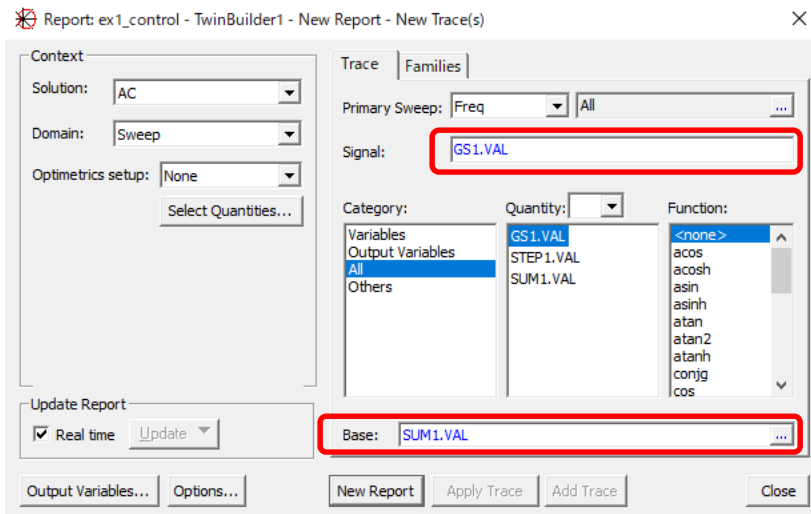
- TR解析を実行して結果を確認します。
 - [Result]>[Create Standard Report]>[Rectangular Plot]をクリックします。



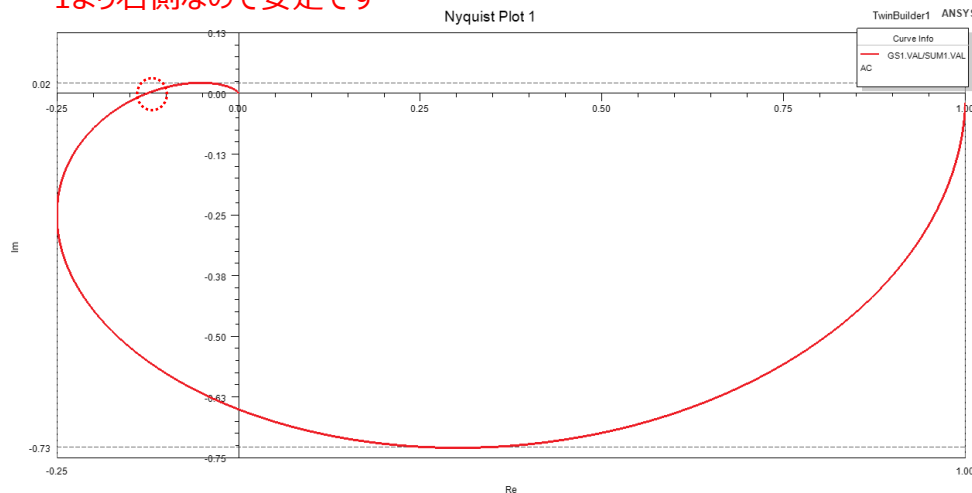
目標値1に対して半分程度で安定(偏差が残っている)

シミュレーションによる確認

- AC解析を実行して結果を確認します。
 - [Result]>[Create Standard Report]>[Nyquist Plot]をクリックします。

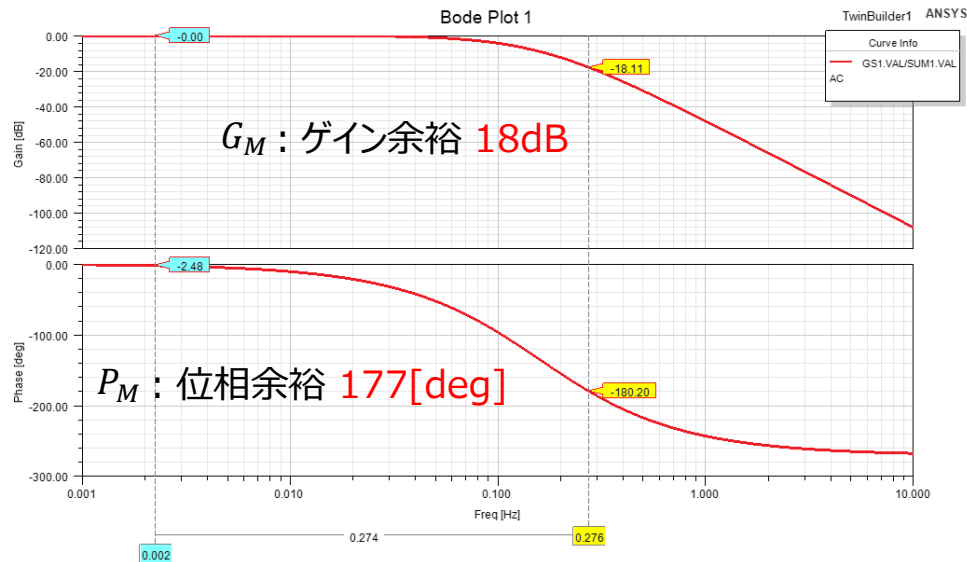
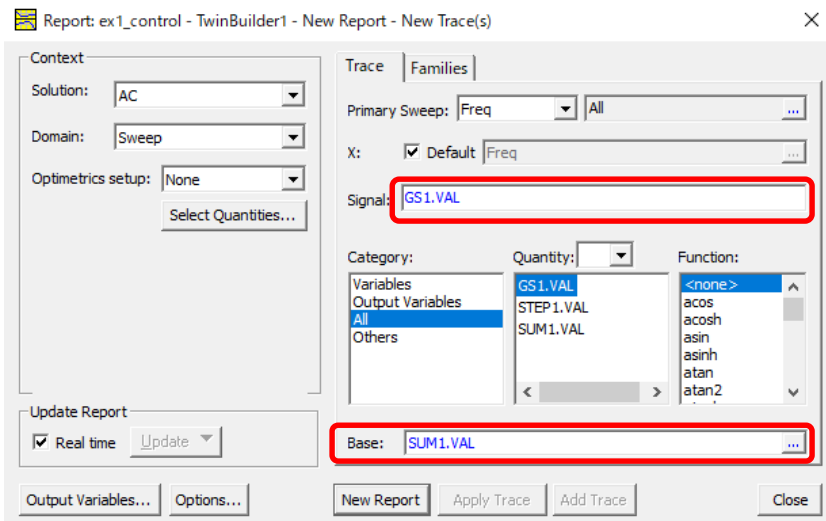


実軸と交差している箇所は
-1より右側なので安定です



シミュレーションによる確認

- AC解析を実行して結果を確認します。
 - [Result]>[Create Standard Report]>[Bode Plot]をクリックします。



シミュレーションによる確認

- コントローラ k_p の値を変更して安定性がどう変化するか確認してみてください.

安定度の目安

プロセス制御の場合

ゲイン余裕が3～10dB , 位相余裕が20[deg]以上

サーボ制御の場合

ゲイン余裕が10～20dB , 位相余裕が40～60[deg]



第4章 伝達関数とPID制御

PID制御

- PIDコントローラ $C(s)$ には、以下の3つの動作が含まれます。

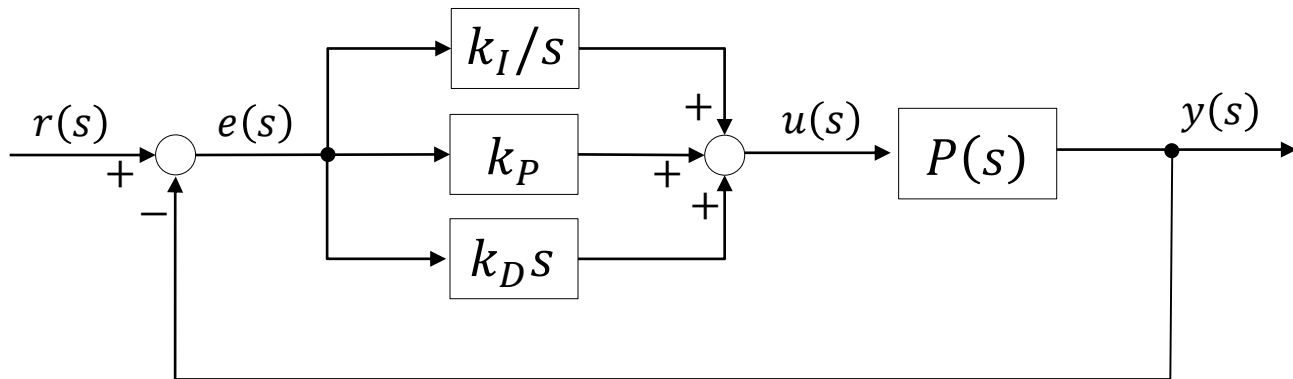
比例動作(P動作)：制御量 $y(t)$ と目標値 $r(t)$ との偏差 $e(t) = r(t) - y(t)$ が大きければ操作量 $u(t)$ を大きくし、偏差 $e(t)$ が小さければ操作量 $u(t)$ を小さくする。

(偏差 $e(t)$ の現在の情報を反映)

積分動作(I動作)：偏差 $e(t)$ の積分値を反映(偏差 $e(t)$ の過去の情報を反映)するような制御を行い**定常偏差を改善**する。

微分動作(D動作)：偏差 $e(t)$ の変化量(微分値)を反映するような制御を行い、**安定性を改善**する。

PID制御



PID制御による閉ループシステム

PID制御

- 時間領域におけるPIDコントローラを式で表すと,

$$\begin{aligned} u(t) &= k_P \left(e(t) + \frac{1}{T_I} \int_0^t e(\tau) d\tau + T_D \frac{de(t)}{dt} \right) \\ &= k_P e(t) + k_I \int_0^t e(\tau) d\tau + k_D \frac{de(t)}{dt} \end{aligned}$$

ここで k_P : 比例ゲイン, k_I : 積分ゲイン, k_D : 微分ゲイン, T_I : 積分時間, T_D : 微分時間と呼びます. 上記の式をラプラス変換すると

$$\begin{aligned} u(s) &= C(s)e(s) \\ C(s) &= k_P \left(1 + \frac{1}{T_I s} + T_D s \right) = k_P + \frac{k_I}{s} + k_D s \end{aligned}$$

PID制御

- 限界感度法によるPIDパラメータの決定

限界感度法と呼ばれる手法では次のようにしてパラメータの調整を行います。まずPコントローラを用いた予備実験で比例ゲインを徐々に大きくしていき、**安定限界となる比例ゲイン k_{pc}** を調べます。この時の**振動周期 T_c** の値を基にして K_P, K_I, K_D を以下の表のように決定します。

	比例ゲイン K_P	積分ゲイン K_I	微分ゲイン K_D
P制御	$0.5 k_{pc}$	-	-
PI制御	$0.45 k_{pc}$	$1.2 K_P / T_U$	-
PID制御	$0.6 k_{pc}$	$2 K_P / T_U$	$K_P T_U / 8$

シミュレーションによる確認

- 例

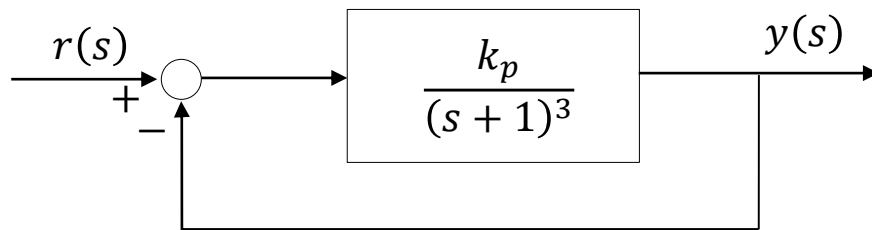
前章で使用した制御対象の伝達関数と

$$P(s) = \frac{1}{(s+1)^3}$$

コントローラの伝達関数を

$$C(s) = k_p, \quad k_p > 0$$

として安定限界までゲインを上げてみます. ※ $k_p = 8$ が安定限界です.



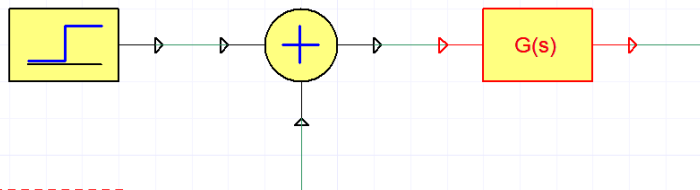
シミュレーションによる確認

- $k_p = 8$ にしたときのシミュレーション結果(安定限界の確認)

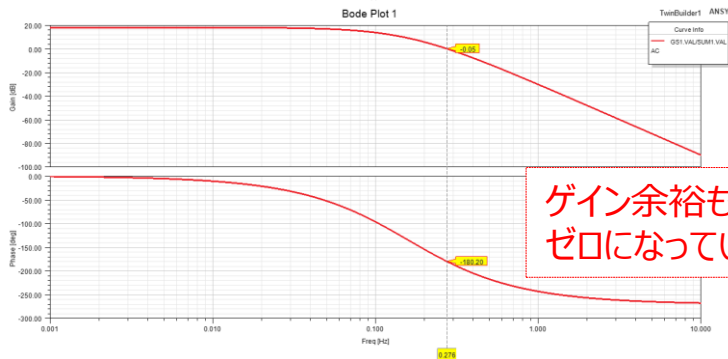
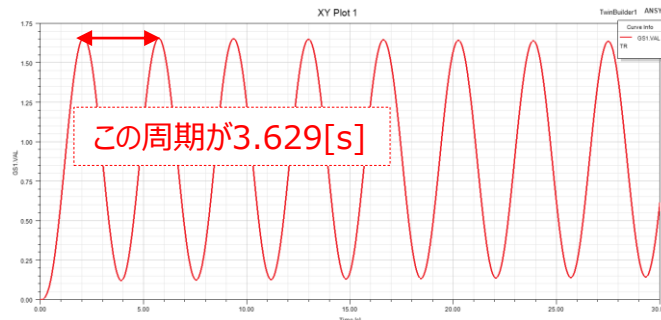
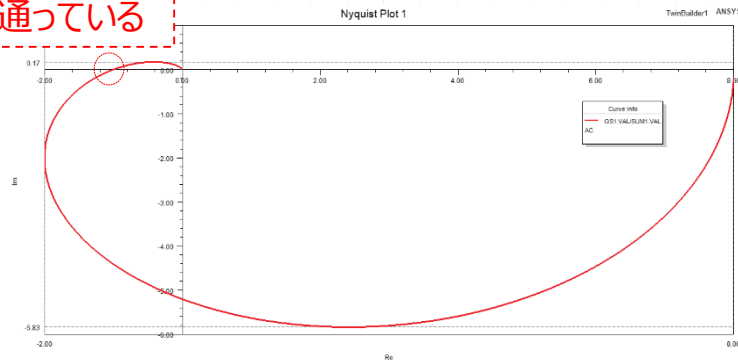
STEP1

SUM1

GS1



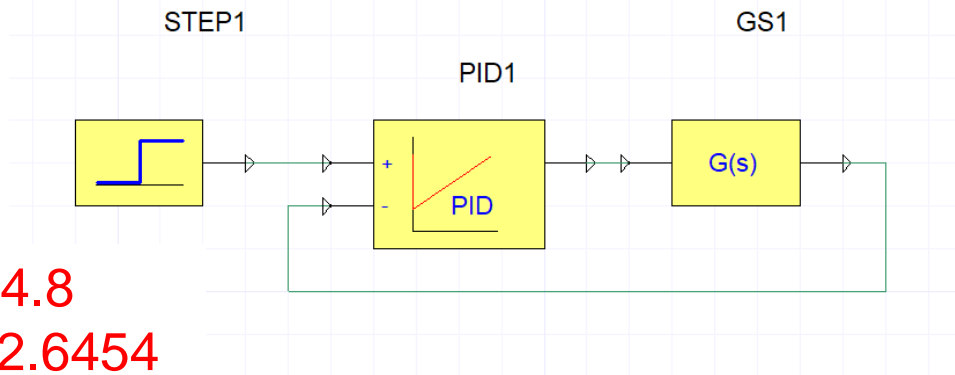
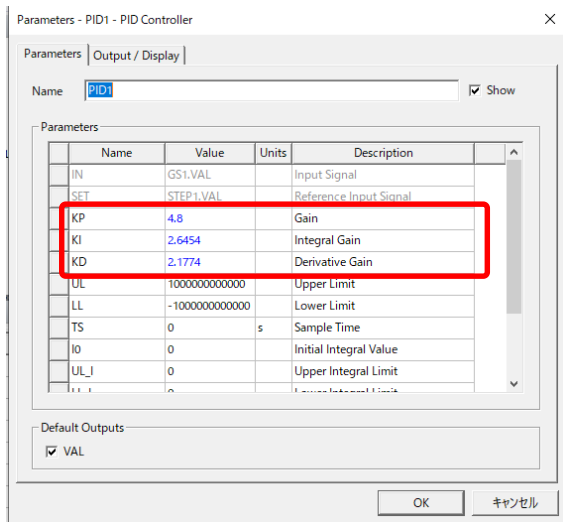
-1を過っている



シミュレーションによる確認

- 限界感度法により求めたPIDパラメータを与えてシミュレーションを実施する.

PIDブロックの設定



KP: 4.8

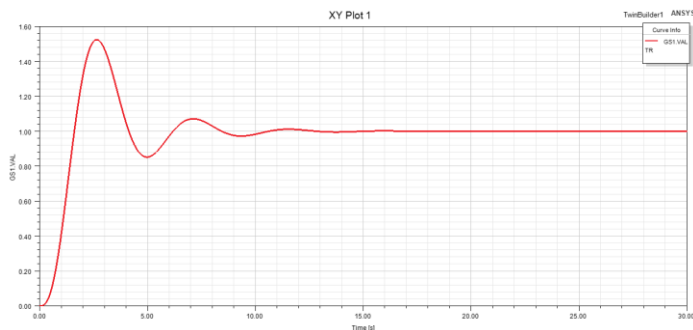
KI : 2.6454

KD: 2.1774

※P40の表を使って求めます

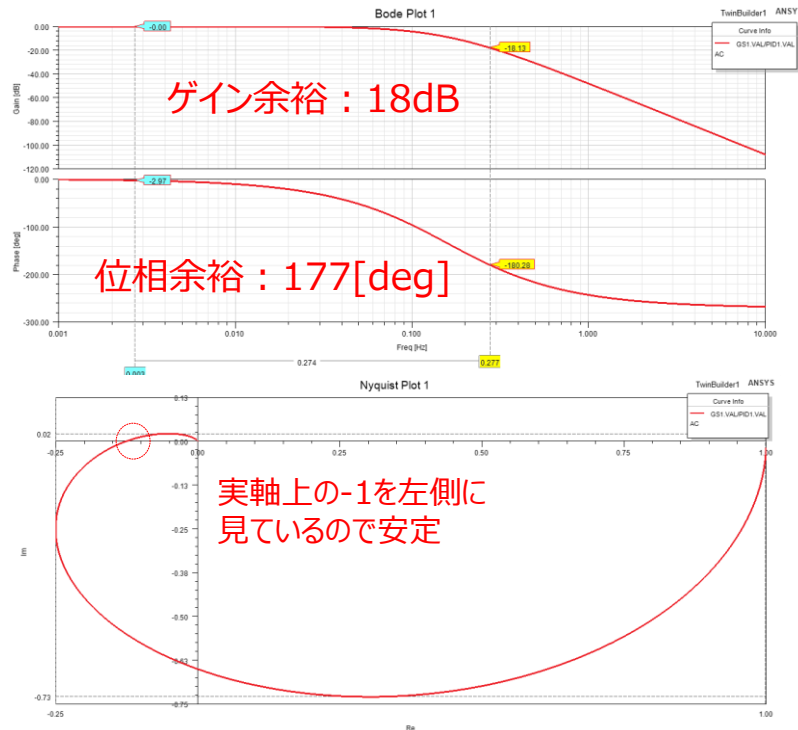
シミュレーションによる確認

時間領域シミュレーション



目標値の1に追従しています

周波数領域シミュレーション





第5章 物理モデリングと制御パラメータの最適化

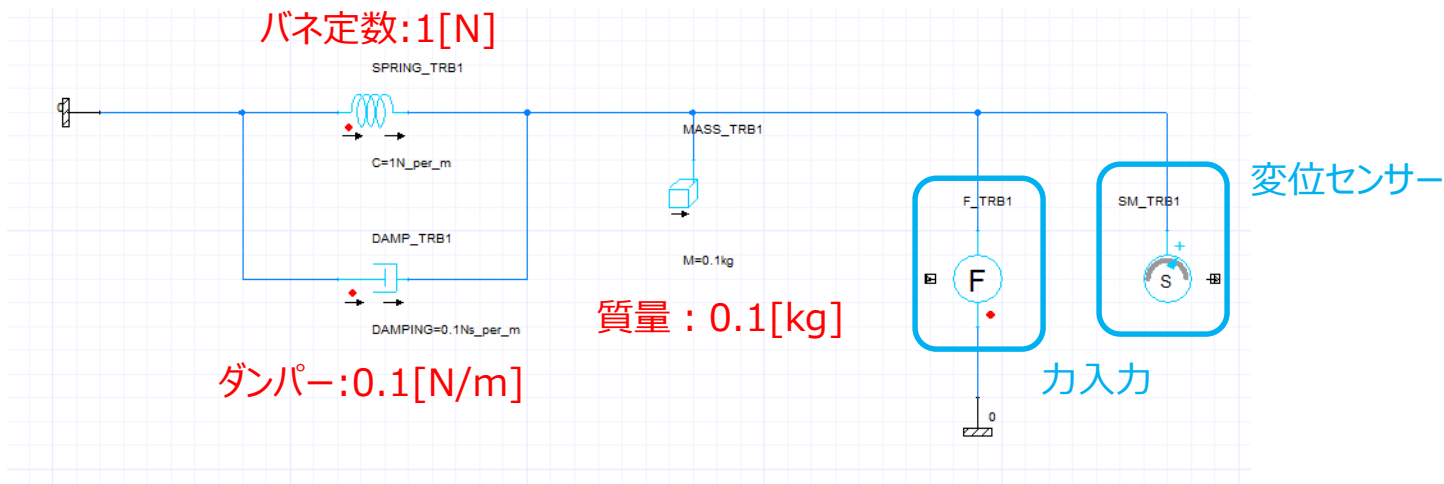
物理モデリング

- 前章までは伝達関数をベースに制御シミュレーションの方法をご紹介してきました。Ansys Twin Builderは伝達関数などの線形モデル以外にも様々な物理領域のシミュレーションが可能です。ここでは最後にバネ-マス-ダンパー系の振動モデルとDCモータを題材にPID制御(PI制御)のパラメータを最適化を使用してチューニングしてみたいと思います。

※この動画では概要の紹介にとどめたいと思いますが別途モデルと操作手順書も用意しておりますのでご希望の方は弊社までご連絡ください。

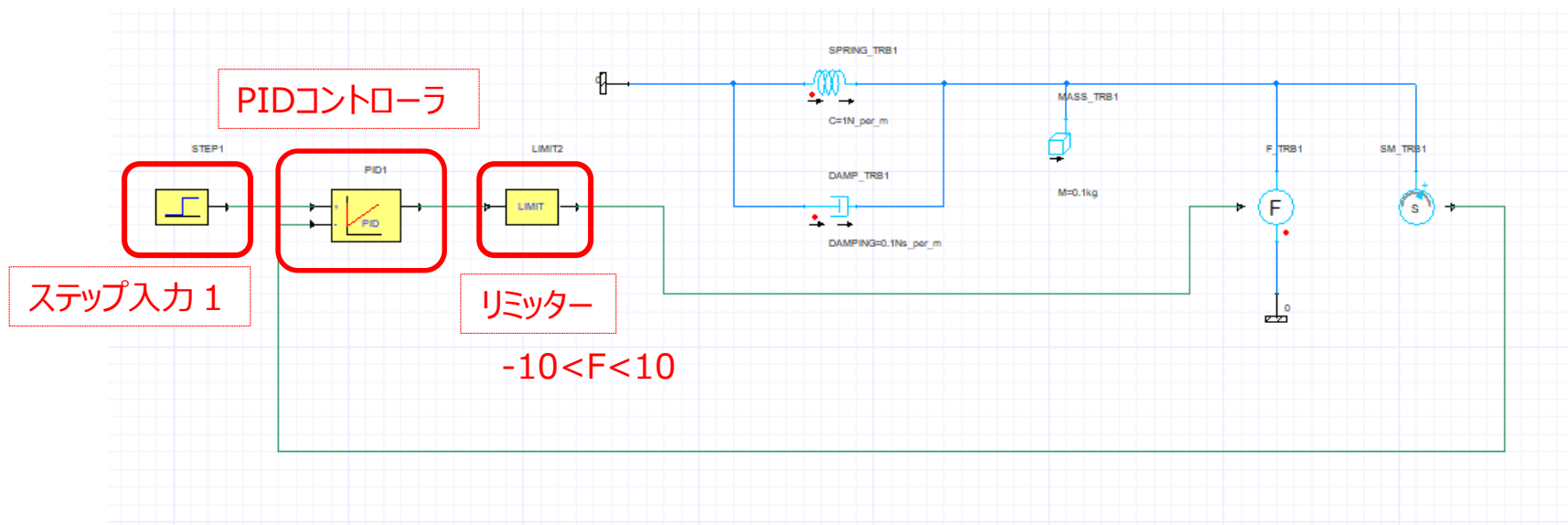
バネ-マス-ダンパーモデルのPID制御

- Ansys Twin Builderで、以下のようなバネ-マス-ダンパーのモデルを作成します。

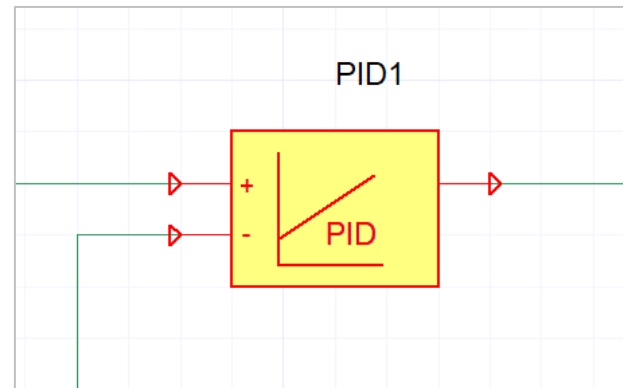
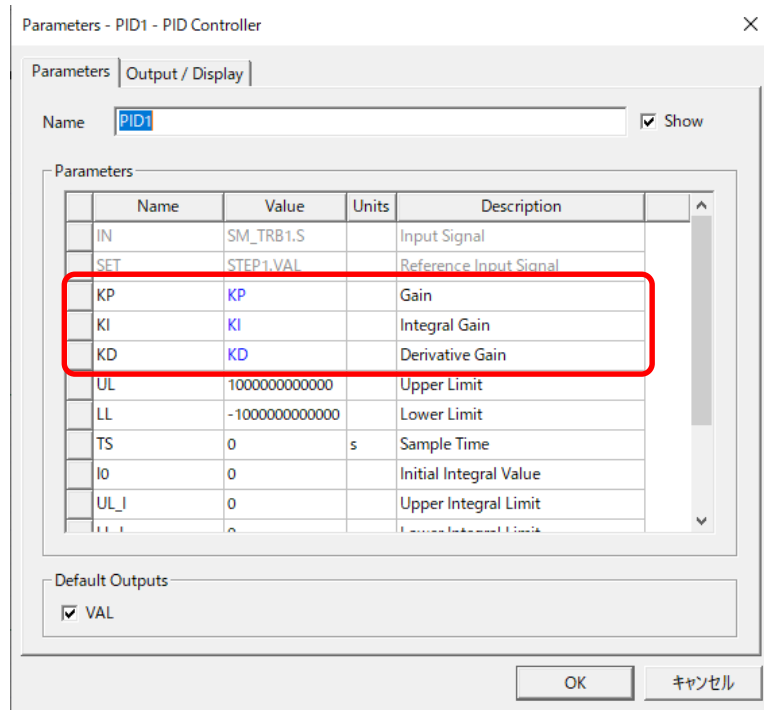


バネ-マス-ダンパーモデルのPID制御

- バネ-マス-ダンパーのモデルにPIDコントローラを接続します.



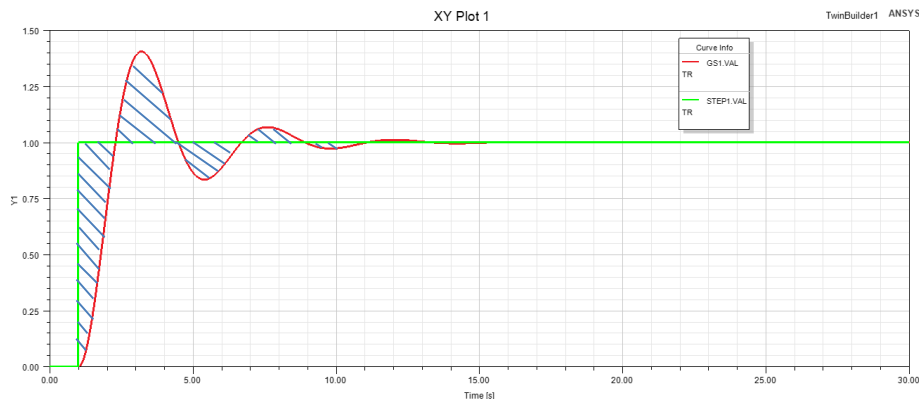
バネ-マス-ダンパーモデルのPID制御



PID制御のパラメータを最適化を適用して
求めるためにTwinBuilderの中の上書き
可能なパラメータとしておきます。

バネ-マス-ダンパーモデルのPID制御

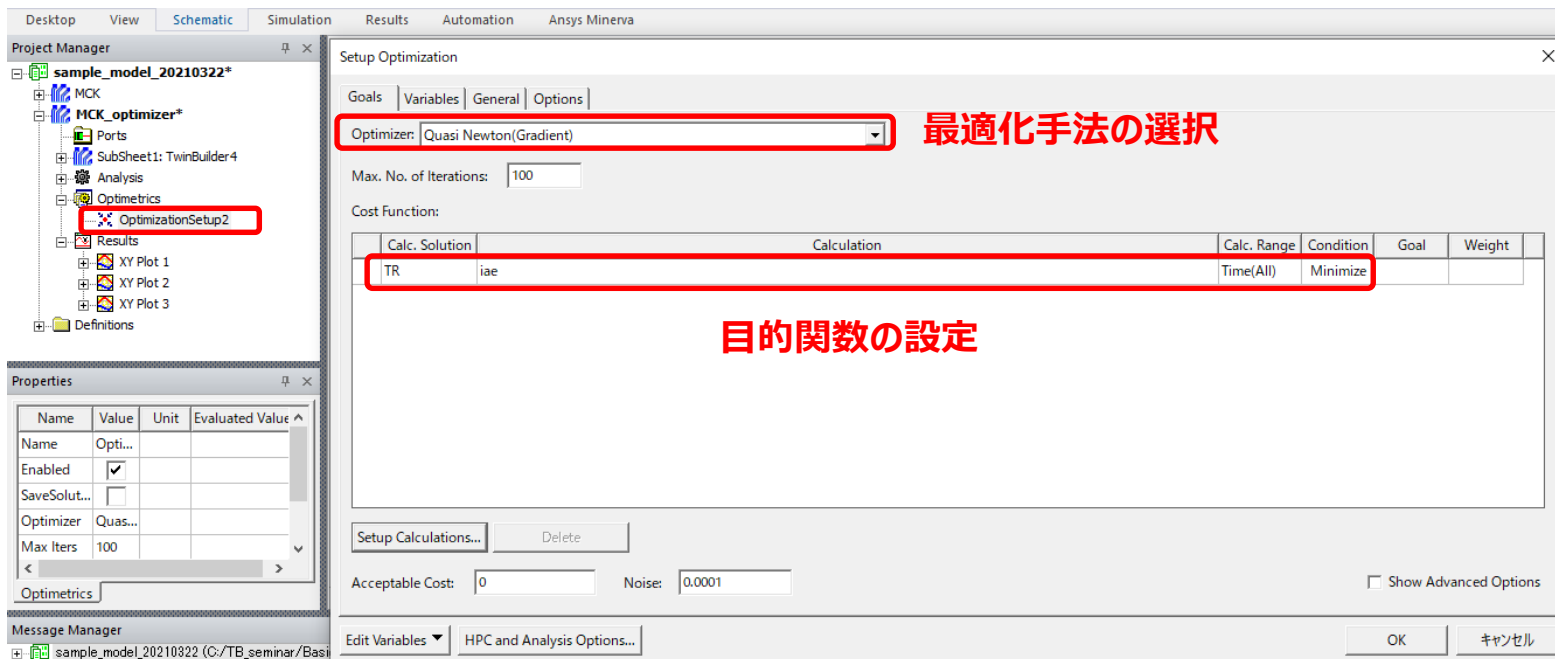
- Ansys Twin Builderの最適化機能を使用して以下のような評価関数(目的関数)を定義します.



左記のグラフの斜線の領域を
最小化するようにPIDパラメータ
を最適化します.

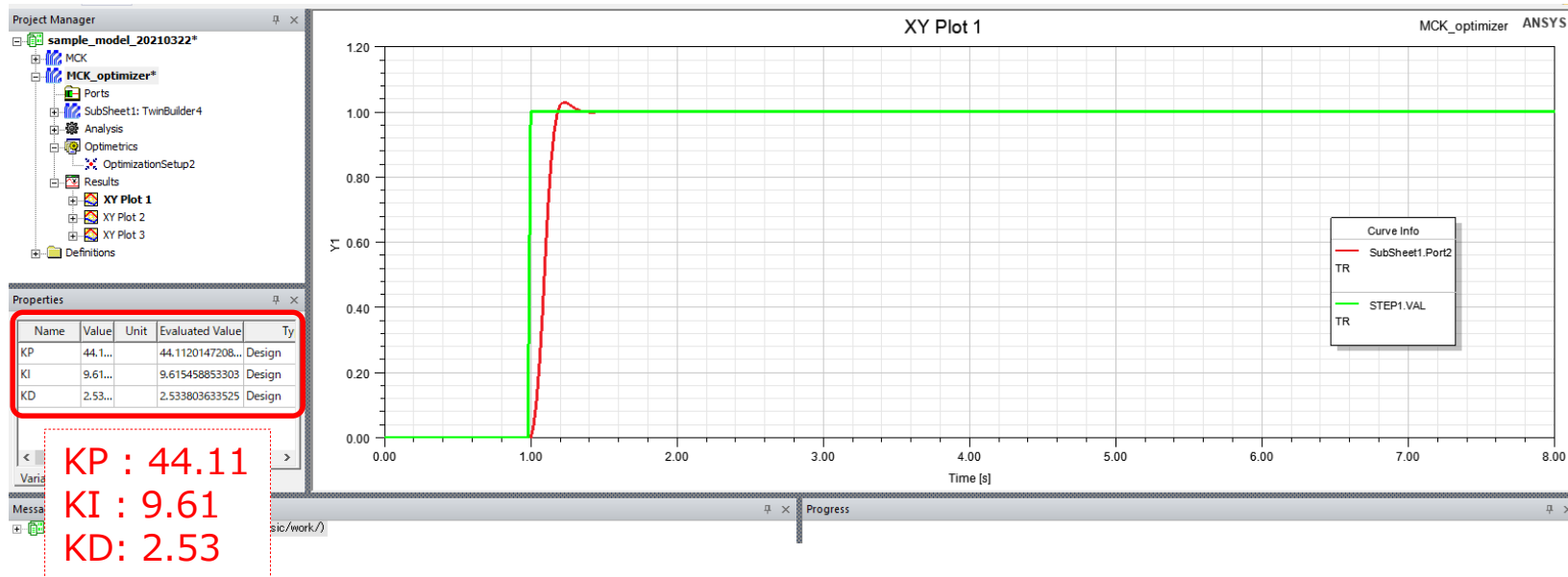
バネ-マス-ダンパーモデルのPID制御

- Optimetrics> OptimizationSetupで最適化の設定をします.



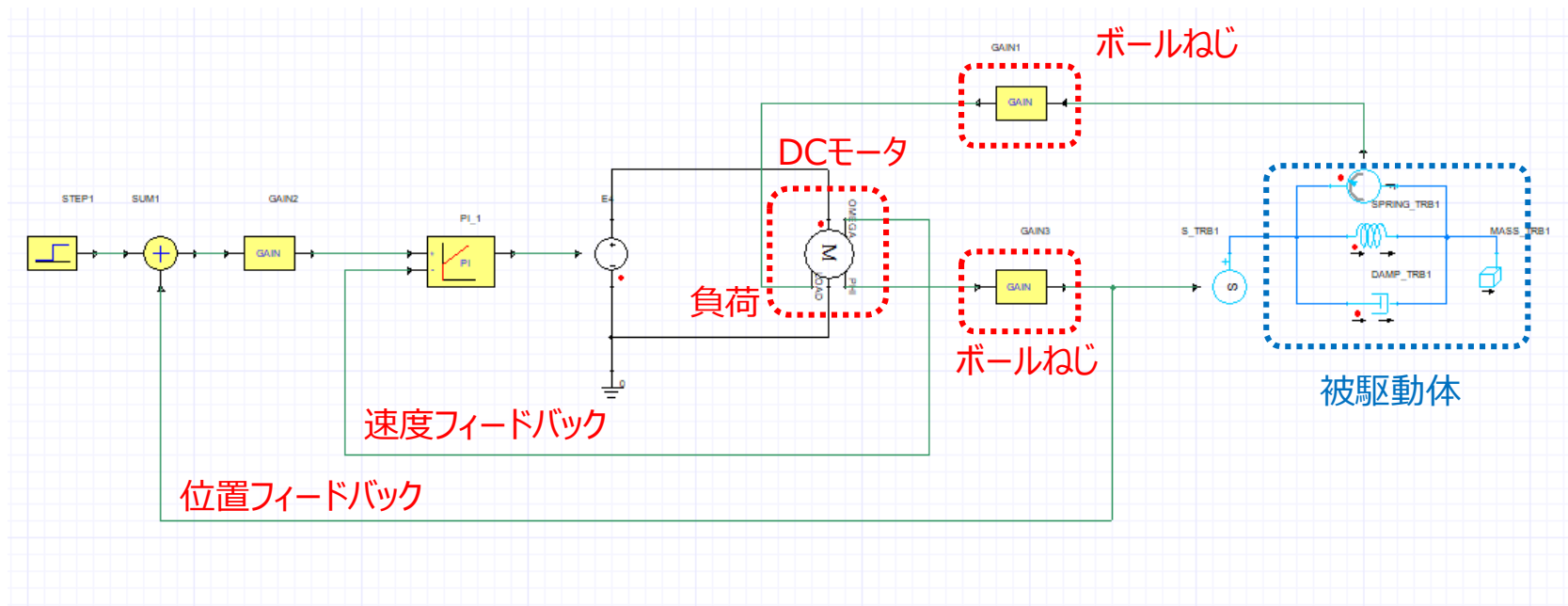
バネ-マス-ダンパーモデルのPID制御

- 最適化計算によって求めたPIDパラメータを使用した計算結果



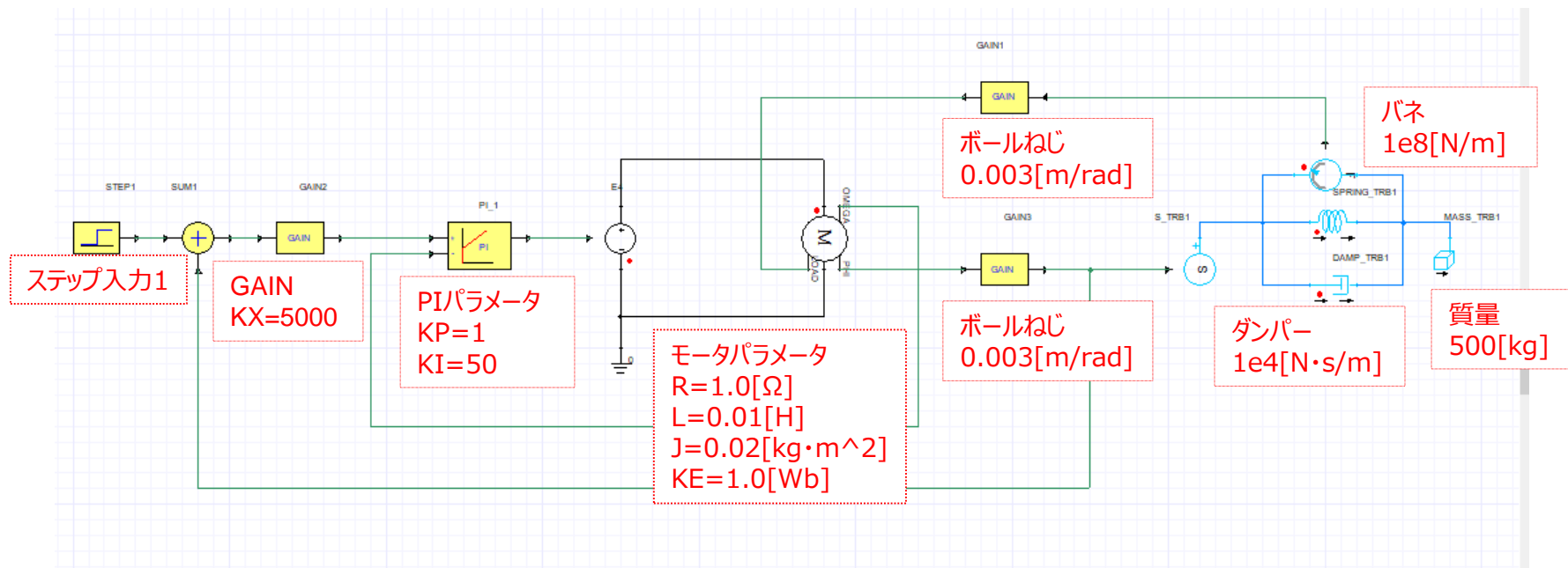
DCモータモデルのPI制御

- DCモータに駆動体を結合して位置決め制御をしてみます。



DCモータモデルのPI制御

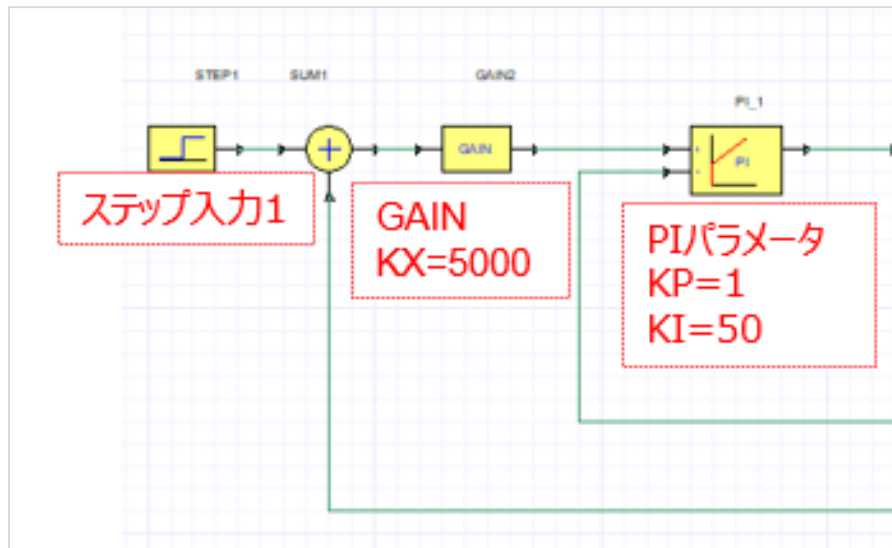
- 各部分のパラメータは以下の通りです.



DCモータモデルのPI制御

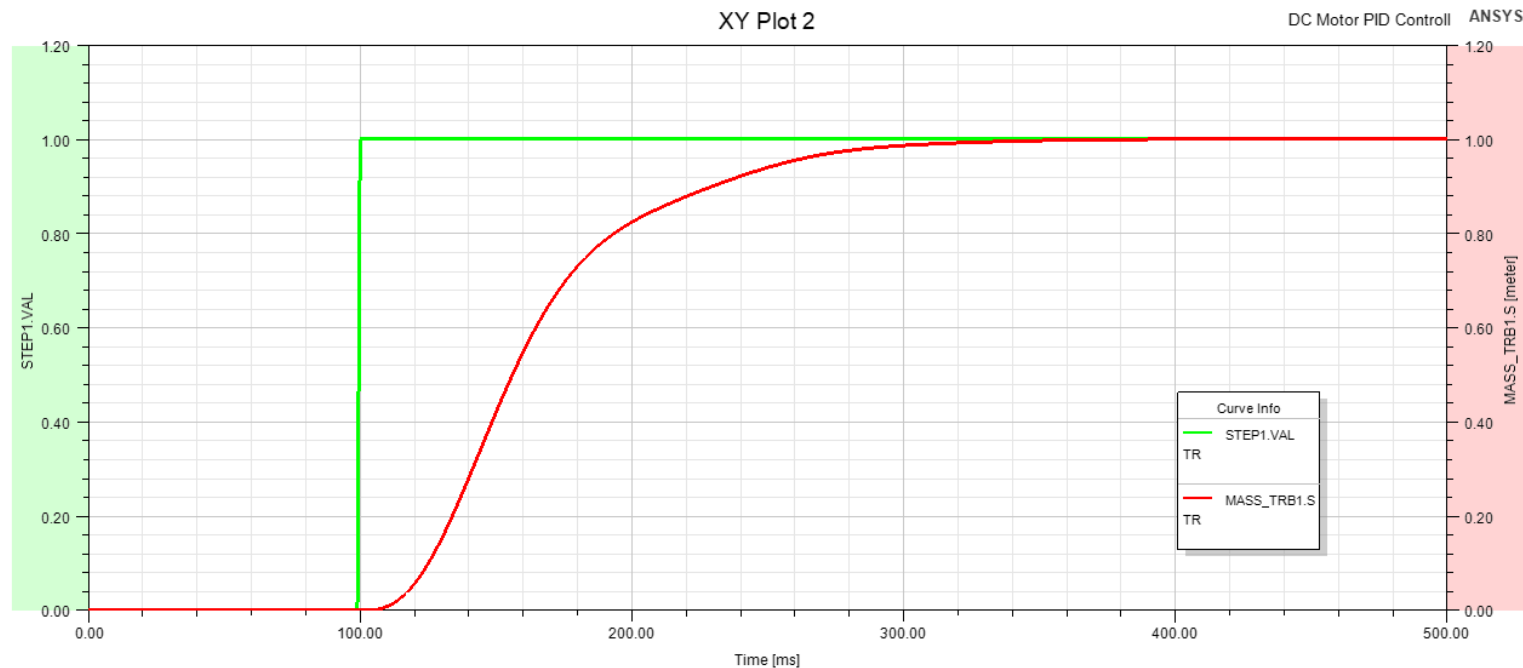
- 最適化を適用するために以下の制御パラメータを上書き可能なパラメータに指定しておきます。

Properties				
Name	Value	Unit	Evaluated Value	Type
KX	5000		5000	Design
KP	1		1	Design
KI	50		50	Design



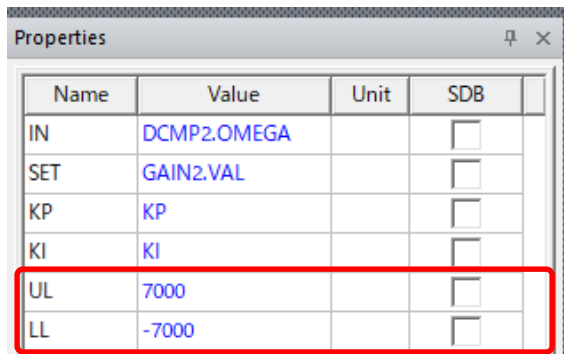
DCモータモデルのPI制御

- 被駆動体の位置のシミュレーション結果

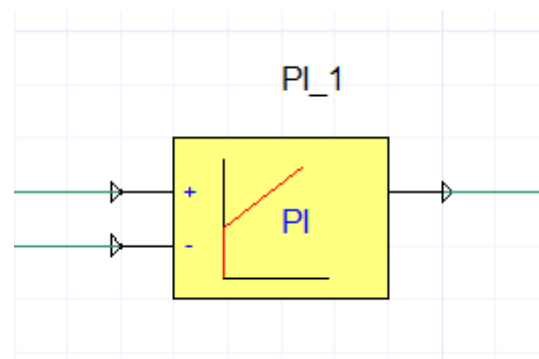


DCモータモデルのPI制御

- 前述のバネ-マス-ダンパーモデルと同様にステップ信号との積分面積を最小にする目的関数を定義して、それを最小化するように最適化の計算を実施してみました。ここではPIのゲイン(KPとKI)を大きくすれば制御性能を上げられると考えられたのでPI制御器に上限と下限を設けて最適化を実施してみました。

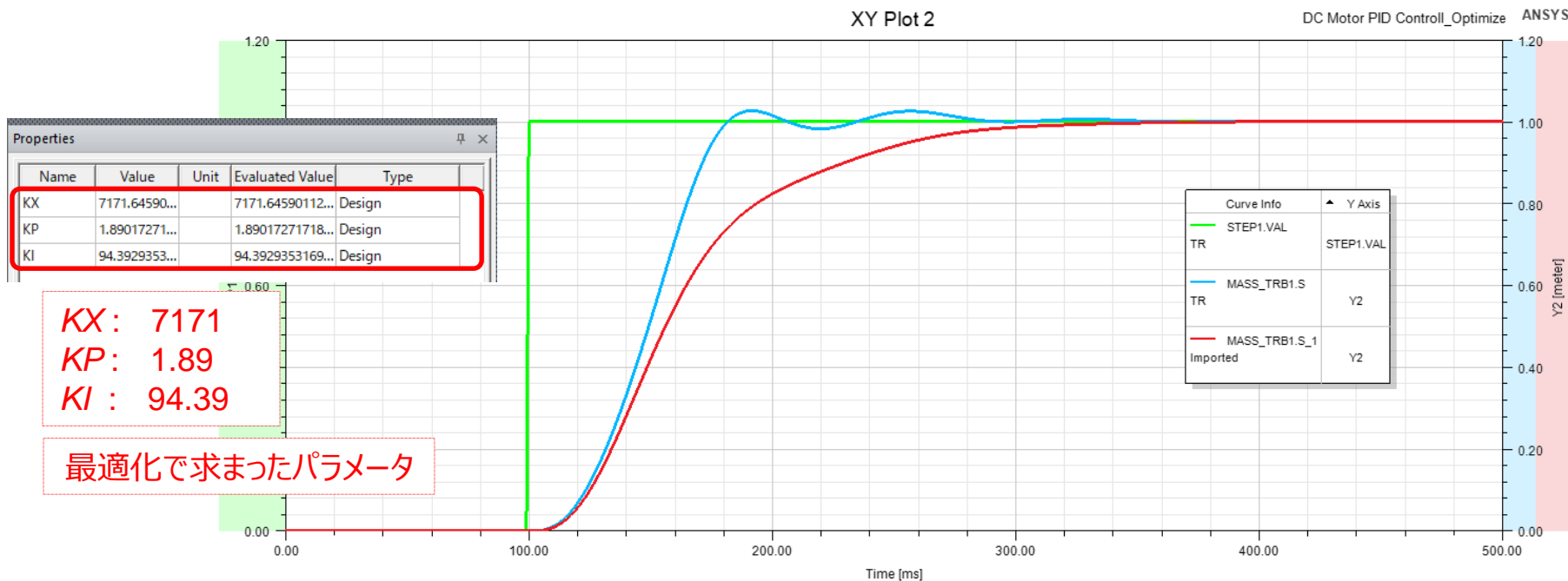


Name	Value	Unit	SDB
IN	DCMP2.OMEGA		<input type="checkbox"/>
SET	GAIN2.VAL		<input type="checkbox"/>
KP	KP		<input type="checkbox"/>
KI	KI		<input type="checkbox"/>
UL	7000		<input type="checkbox"/>
LL	-7000		<input type="checkbox"/>



DCモータモデルのPI制御

- (最適化で求めたパラメータを使用した)被駆動体の位置の結果





サイバネットシステム株式会社
〒101-0022
東京都千代田区神田練塀町3番地 富士ソフトビル