SICE関西支部主催のオンライン勉強会2020

Pythonで学ぶ制御工学

Part1: Pythonの基礎から制御系解析まで

大阪大学
OSAKA UNIVERSITY

南 裕樹

勉強会の概要

流れや雰囲気を体感してください
独学のハードルを下げるのが目的です！本を読んでください！

Part 1

制御工学の基礎
- 制御とは，フィードバック制御，制御系設計

Pythonプログラミングの基礎
- Jupyter Notebook の使い方
- Pythonプログラミング　★ Python演習 1

制御系設計のためのモデル
- 伝達関数モデル，ブロック線図，時間応答，周波数応答　★ Python演習 2

Part 2

制御系設計のためのモデル（復習）
- 伝達関数モデル，ブロック線図，時間応答，周波数応答

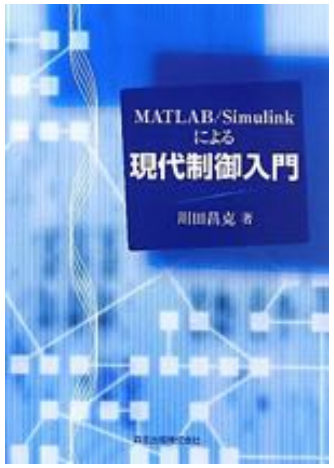
伝達関数モデルを用いた制御系設計
- 閉ループ系の設計仕様，PID制御，モデルマッチング　★ Python演習 3

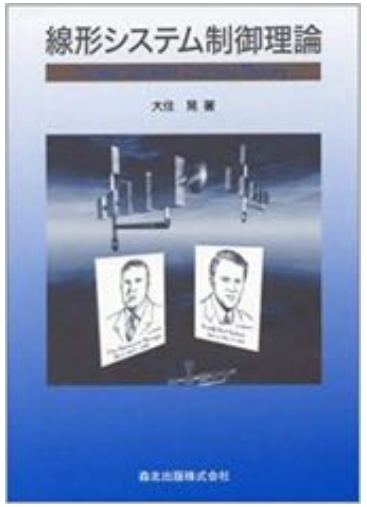
Part 3

ループ整形による制御系設計
- 開ループ系の設計仕様，位相進み・遅れ補償　★ Python演習 4

状態空間モデルを用いた制御系設計
- 状態空間モデル，状態フィードバック　★ Python演習 5

参考図書





導入にもってこい

押さえておけば安心

数学的にしっかり勉強

補助資料の紹介

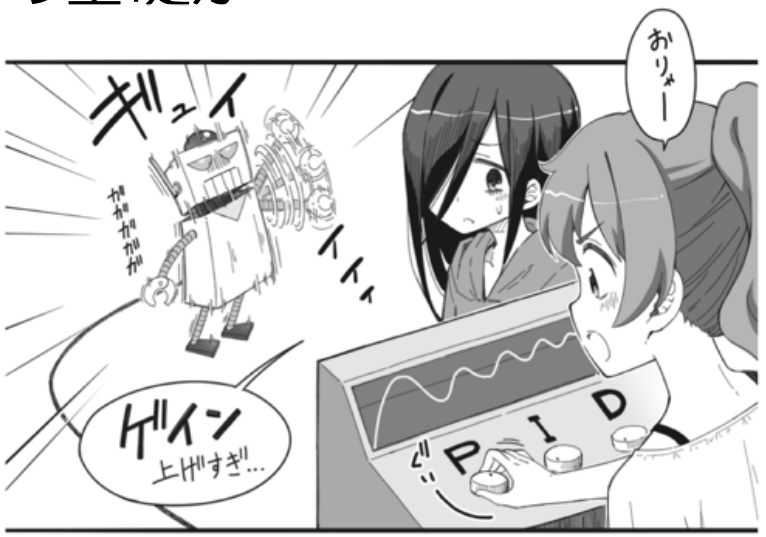
「Python × 制御工学」の日本初の本

古典制御，現代制御，ロバスト制御の基礎が
ギュッとつまっている

数学的な説明は少なめ

対話形式の説明やイラストを配置

サポートページには，
サンプルコードが公開されている
→ Python & MATLABコード



- 2,600 円＋税
- A5 272頁
- 2019/05/22 発行　Kindle版もあります

補助資料の紹介

第1章　制御とは
第2章　Pythonの基礎
} 準備

第3章　制御のためのモデル ---------- モデリング

第4章　制御対象の振る舞い ---------- 解析

第5章　閉ループ系に注目した制御系設計
第6章　開ループ系に注目した制御系設計
第7章　アドバンストな制御系設計
} 設計

付録　数学の補足　（オブザーバ，混合感度問題，離散化）

※微分積分や線形代数の基本計算ができることが前提です
※プログラミングや力学・電気回路の素養があると有利です
※古典制御と現代制御の内容をあえて混ぜて書いています

なぜPythonで制御系設計？

スクリプト言語（インタープリタ型）⇔ コンパイラ型

変数の型宣言が必要ない

インデントが構文規則の一部

- ✓ ブロック（for や if などの範囲）をインデントで表現
  - → **誰でも綺麗なコードが書ける**

C

```c
int main(void)
{
    int i = 0;

    for(i=0; i<5; i++){
        printf("%d¥n", i);
    }
    return 0;
}
```

Python

```python
for i in range(0, 5, 1):
    print(i)
```

なぜPythonで制御系設計？

スクリプト言語（インタープリタ型）⇔ コンパイラ型

変数の型宣言が必要ない

インデントが構文規則の一部

- ✓ ブロック（for や if などの範囲）をインデントで表現
  - → **誰でも綺麗なコードが書ける**

C

```c
int main(void)
{
int i = 0;
for(i=0; i<5; i++){
printf("%d¥n", i);
}
return 0;
}
```

- ✓ これでも実行可
  でも，汚い

Python

```python
for i in range(0, 5, 1):
    print(i)
```

- ✓ 可読性が高い
- ✓ メンテナンス性に優れる

なぜPythonで制御系設計？

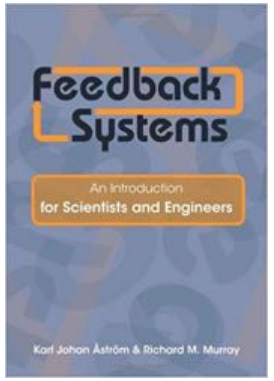
演習環境がタダで簡単に構築できる！

- Anacondaをダウンロード → インストール＋α
- Google Colab （クラウドサービス）を利用+α → iPadもOK

Pythonユーザが多く，やりたいことは大体できる

- ライブラリが豊富（数値計算，数式処理，信号処理，etc）
- web上にいろいろな情報が公開されていて便利

制御工学ライブラリが整備されている



- R. M. Murray のグループが開発（信頼できそう）
- Matlabのコマンドとほぼ同じ → 将来的にMatlabに移行可

今日の目標

Level☆☆★

- 制御とはなにかを説明できる
- 伝達関数モデルのステップ応答のグラフをPythonで描ける
- 伝達関数モデルの周波数特性のグラフをPythonで描ける

Level☆★★

- 伝達関数モデルを作ることができる
- １次遅れ系，２次遅れ系の特性を理解できる

Level★★★

- 設計モデルの種類とそれらの関係を理解できる
- ステップ応答や周波数応答を手計算で求めることができる

勉強会の概要

制御工学の基礎
- 制御とは，フィードバック制御，制御系設計

Pythonプログラミングの基礎
- Jupyter Notebook の使い方
- Pythonプログラミング　★ Python演習 1

制御系設計のためのモデル
- 伝達関数モデル，ブロック線図，時間応答，周波数応答　★ Python演習 2

Part 2

制御系設計のためのモデル（復習）
- 伝達関数モデル，ブロック線図，時間応答，周波数応答

伝達関数モデルを用いた制御系設計
- 閉ループ系の設計仕様，PID制御，モデルマッチング　★ Python演習 3

Part 3

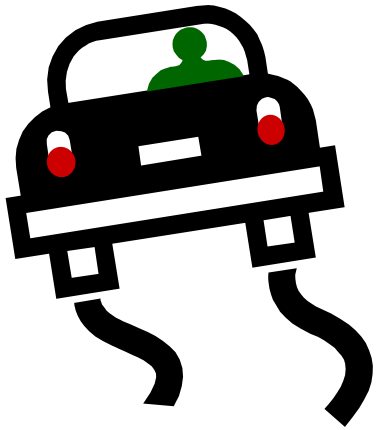
ループ整形による制御系設計
- 開ループ系の設計仕様，位相進み・遅れ補償　★ Python演習 4

状態空間モデルを用いた制御系設計
- 状態空間モデル，状態フィードバック　★ Python演習 5

制御とは

「制御」の定義

「**注目している対象物**に属する**注目している状態**が，なんらかの**目標とする状態**になるように，その対象物に**操作を加える**行為」を制御という



自動車の例（手動制御）

注目している対象物：自動車
注目している状態　：自動車の進行方向
目標とする状態　　：道路にそって走行する
操作を加える　　：ハンドルを操作

制御とは

「制御」の定義

「**注目している対象物**に属する**注目している状態**が，なんらかの**目標とする状態**になるように，その対象物に**操作を加える**行為」を制御という

セグウェイの例（自動制御）

注目している対象物：セグウェイ
注目している状態　：姿勢
目標とする状態　　：まっすぐ立つ
操作を加える　　：車輪モータを操作

身のまわりの制御

安心・安全・快適な生活は制御のおかげ

モノの”知能化”を進めるには制御が必要！

モノの**動き**を
デザインする！

＝

制御工学

＝

応用数学

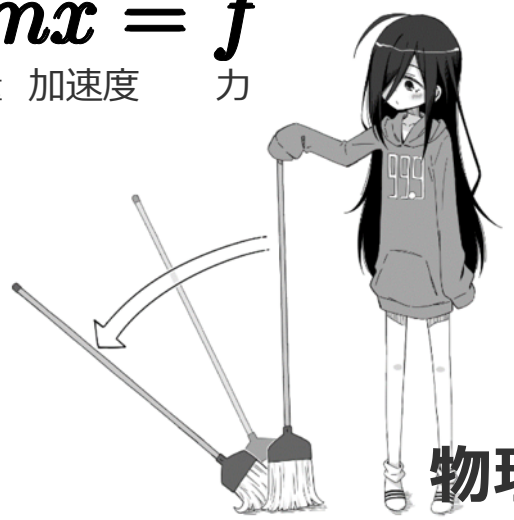
制御“工学”

「制御」の定義

「**注目している対象物**に属する**注目している状態**が，なんらかの**目標とする状態**になるように，その対象物に**操作を加える**行為」を制御という

$$m\ddot{x} = f$$

質量　加速度　　力



物理学（自然法則）

$$m\ddot{x} = u$$

制御入力

主観まみれ

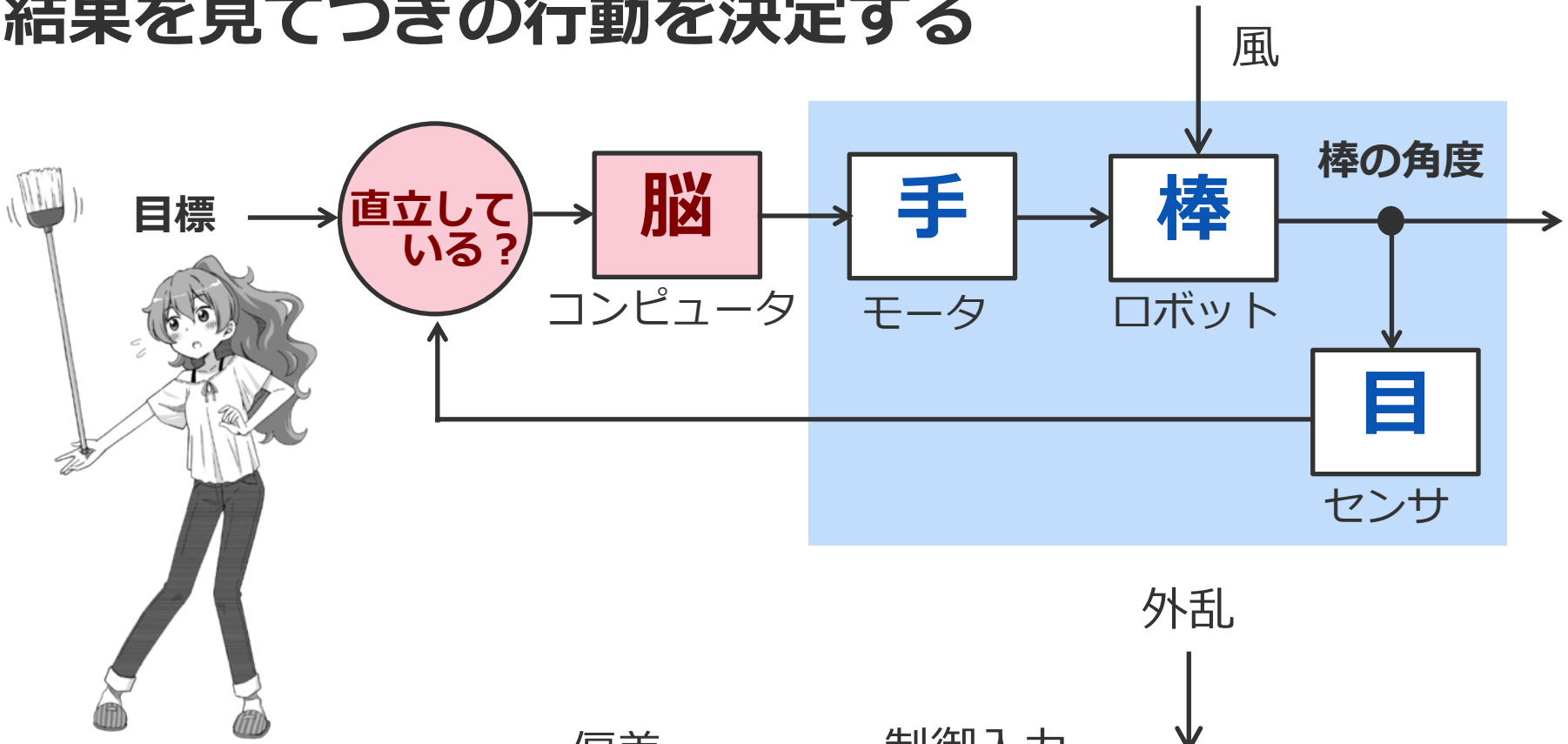
制御入力の設計

- 試行錯誤の設計は良くない
- **設計思想を明確にする必要がある**

人間の物理学

フィードバック制御

結果を見てつぎの行動を決定する



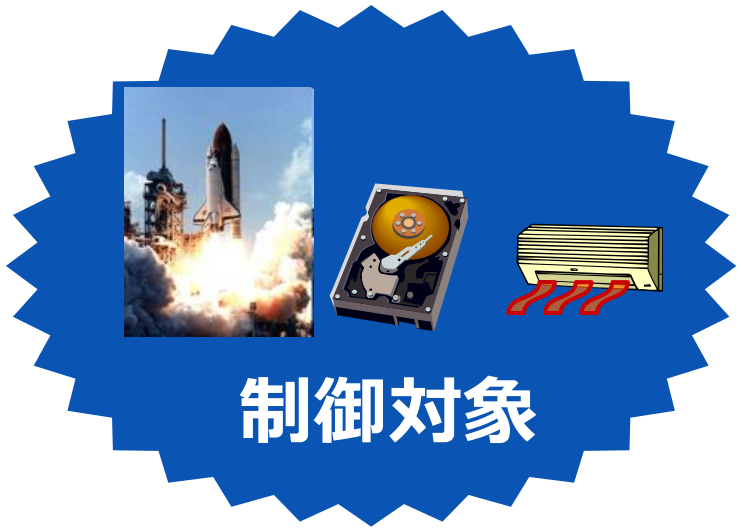
ネガティブフィードバック

制御系設計の流れ

制御目的

定式化

制御仕様



制御対象

モデル化

紙の上の世界

数学モデル
対象の本質をとらえる

評価

設計

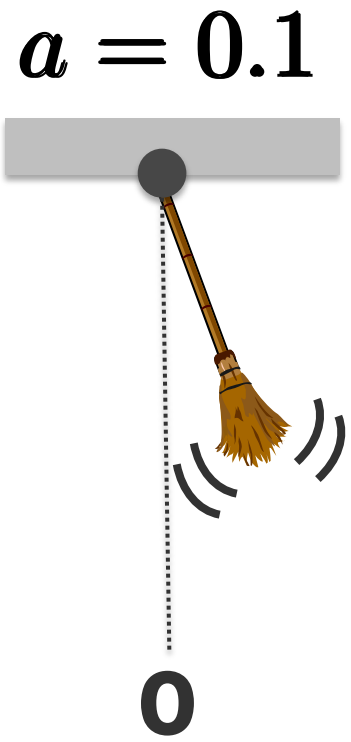
制御則
目標状態にする方法

実装

制御器

実現

安定な制御対象



振子

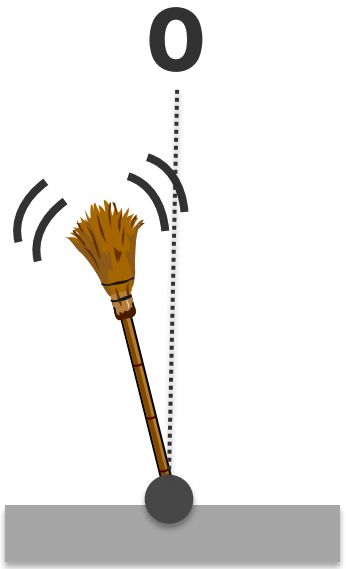
シンプルなモデル（漸化式）

$$x_{k+1} = ax_k \qquad x_0 = 1$$

$$x_1 = 0.1 \times x_0 = 0.1$$
$$x_2 = 0.1 \times x_1 = 0.01$$
$$x_3 = 0.1 \times x_2 = 0.001$$
$$\vdots$$
$$x_\infty = 0$$

0 に収束 ＝ **安定！**

不安定な制御対象



$a = 2.0$

振子

シンプルなモデル（漸化式）

$$x_{k+1} = ax_k \qquad x_0 = 1$$

$$x_1 = 2 \times x_0 = 2$$
$$x_2 = 2 \times x_1 = 4$$
$$x_3 = 2 \times x_2 = 8$$
$$\vdots$$
$$x_\infty = \infty$$

∞に発散 ＝ **不安定！**

制御の力で安定化する＝制御系設計

不安定なものを安定化する

$a = 2.0$

- 手を動かす
- 棒を見ながら
- まっすぐ
  立つように

$$x_{k+1} = ax_k \quad \leftarrow \text{制御系設計（設計モデル）}$$

$$x_{k+1} = ax_k + u_k$$

制御

フィードバック制御！

$$u_k = fx_k \quad \leftarrow \text{制御系設計（制御器の構造）}$$

$$x_{k+1} = (a+f)x_k$$

$$f = -1.9 \quad \leftarrow \text{制御系設計（チューニング）}$$

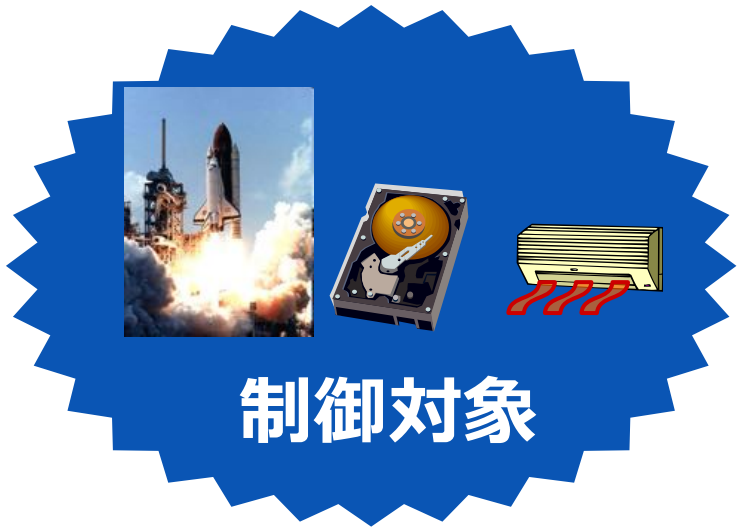
$$x_{k+1} = 0.1 \times x_k$$

制御系設計の流れ

制御目的

定式化

制御仕様



制御対象

モデル化

紙の上の世界

数学モデル

対象の本質をとらえる

評価

設計

制御則

目標状態にする方法

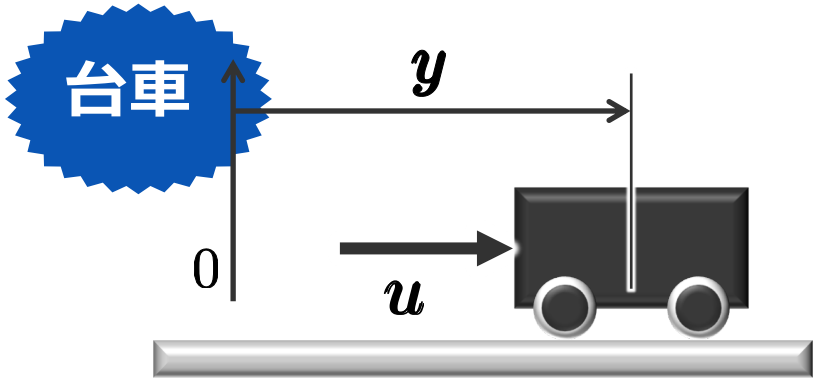
実装

制御器

実現

制御系設計の例題

台車を目標の位置に到達させることを考える



制御目的

y を目標位置 0 に到達させる

→ 適切な u を生成する **制御器**

台車の**数学モデル**（質量＝１，摩擦なし）：$\ddot{y} = u$

【設計条件】**制御則**として，$u(t) = -k_1 y(t) - k_2 \dot{y}(t)$ を考える

パラメータは　k_1，k_2 の二つ

→ $\ddot{y} + k_2 \dot{y} + k_1 y = 0$ となり，解は，$y(t) = C_1 e^{\lambda_1 t} + C_2 e^{\lambda_2 t}$

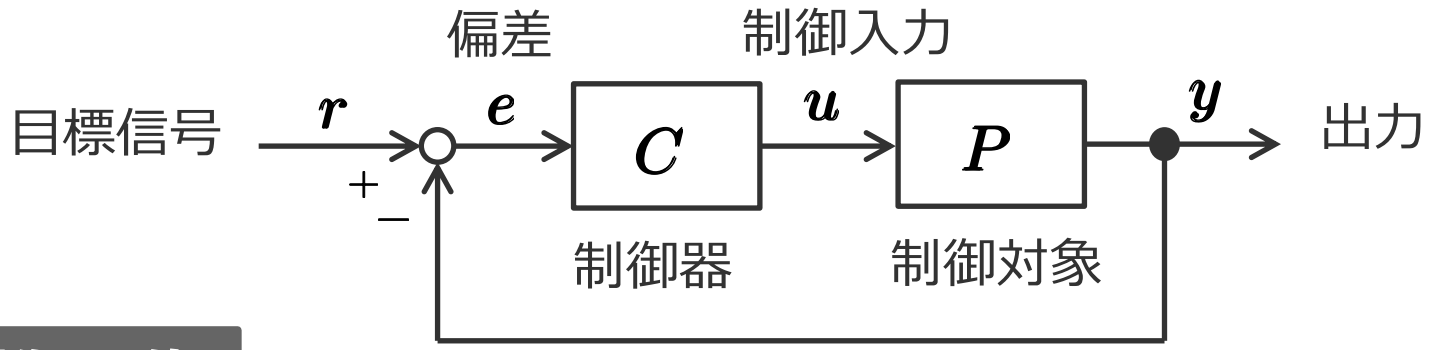
ただし，λ_1，λ_2 は $s^2 + k_2 s + k_1 = 0$ の根である

【設計方程式】λ_1，λ_2 の実部が負になる k_1，k_2 を求める

$$s = \frac{-k_2 \pm \sqrt{k_2^2 - 4k_1}}{2}$$

$k_1 > 0, \quad k_2 > 0$

制御系設計の例題



制御目的

制御対象の出力 y が目標信号 r に定常偏差なく追従する

➡ y が r に追従するような，安定かつロバストな制御系（制御器 C ）を設計せよ

制御仕様

- **設計条件：**制御器の構造，パラメータ

  Ex.) PID制御，位相進み・遅れ，状態フィードバック etc.

  $$C(s) = k_P + \frac{k_I}{s} + k_D s \qquad C : u(t) = Kx(t)$$

- **設計方程式とその求解方法**

  Ex.) 特性多項式が安定，オーバーシュートが10% etc.

勉強会の概要

制御工学の基礎
- 制御とは，フィードバック制御，制御系設計

Pythonプログラミングの基礎
- Jupyter Notebook の使い方
- Pythonプログラミング　★ Python演習 1

制御系設計のためのモデル
- 伝達関数モデル，ブロック線図，時間応答，周波数応答　★ Python演習 2

Part 2

制御系設計のためのモデル（復習）
- 伝達関数モデル，ブロック線図，時間応答，周波数応答

伝達関数モデルを用いた制御系設計
- 閉ループ系の設計仕様，PID制御，モデルマッチング　★ Python演習 3

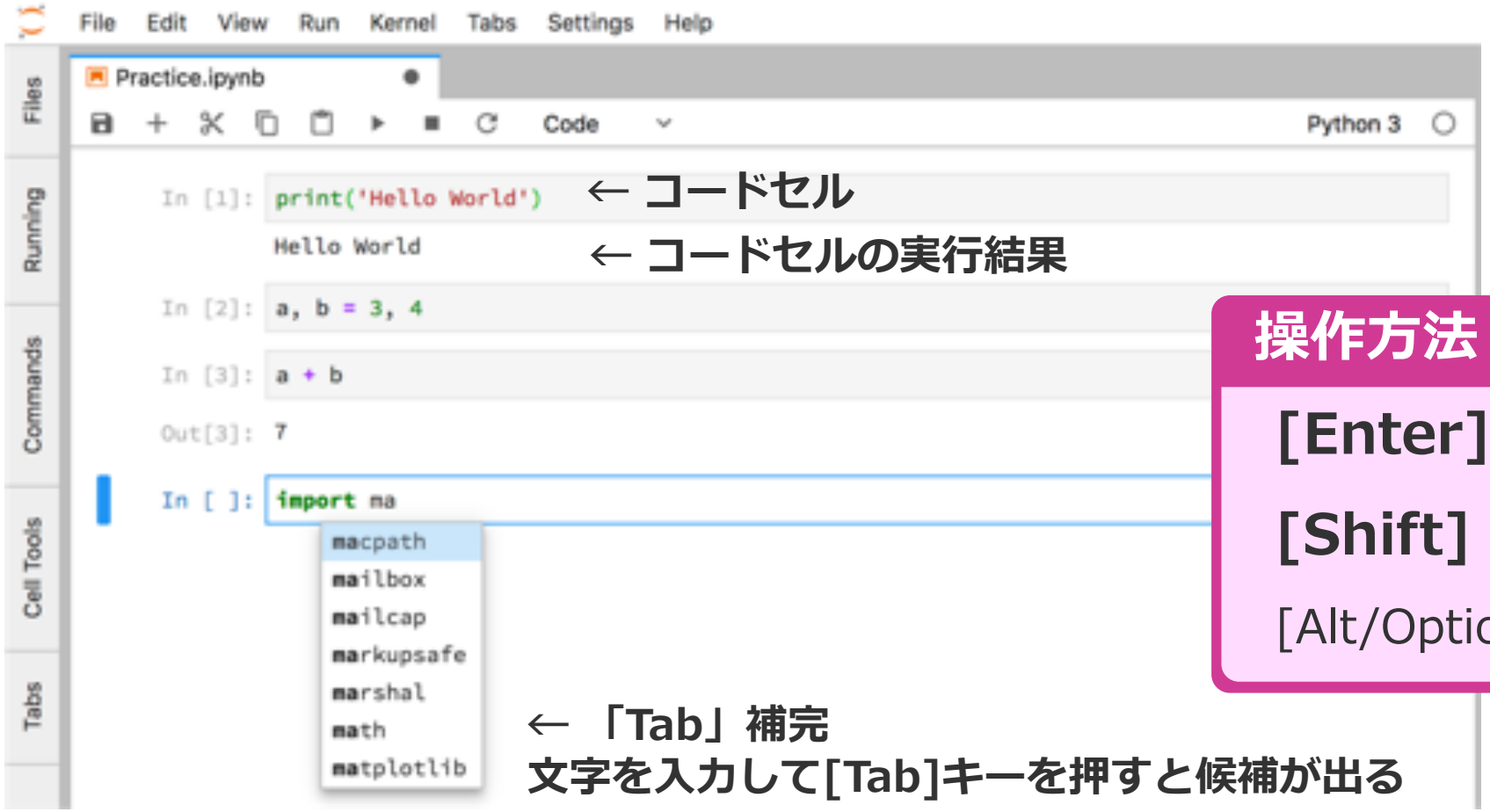
Part 3

ループ整形による制御系設計
- 開ループ系の設計仕様，位相進み・遅れ補償　★ Python演習 4

状態空間モデルを用いた制御系設計
- 状態空間モデル，状態フィードバック　★ Python演習 5

Jupyter Notebook

- ブラウザ上で使えるオープンソースのコーディング環境
- 対話形式で開発を進めることができる　　**修正しての再実行も可能！**



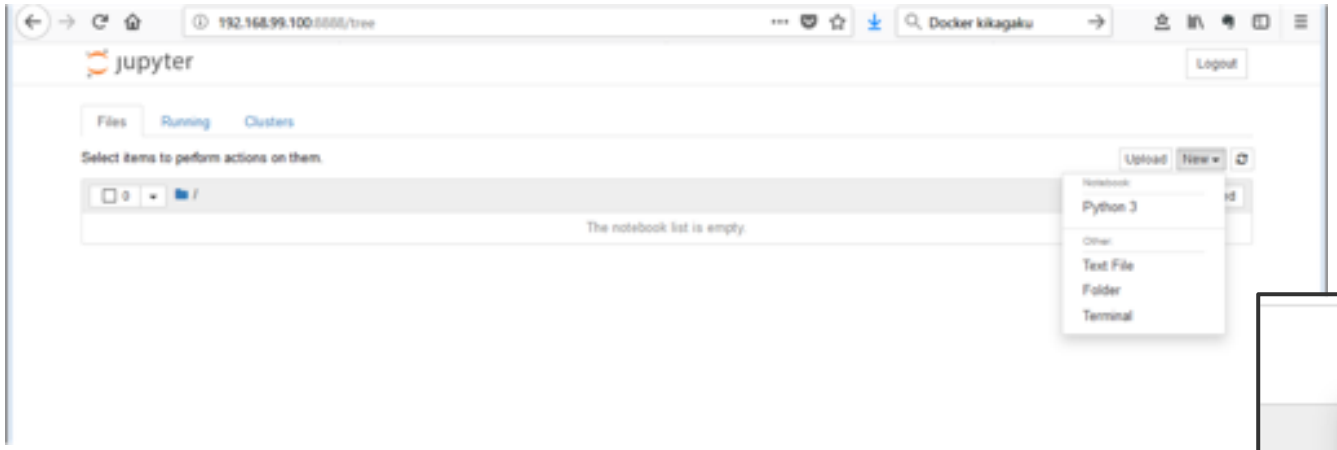
操作方法

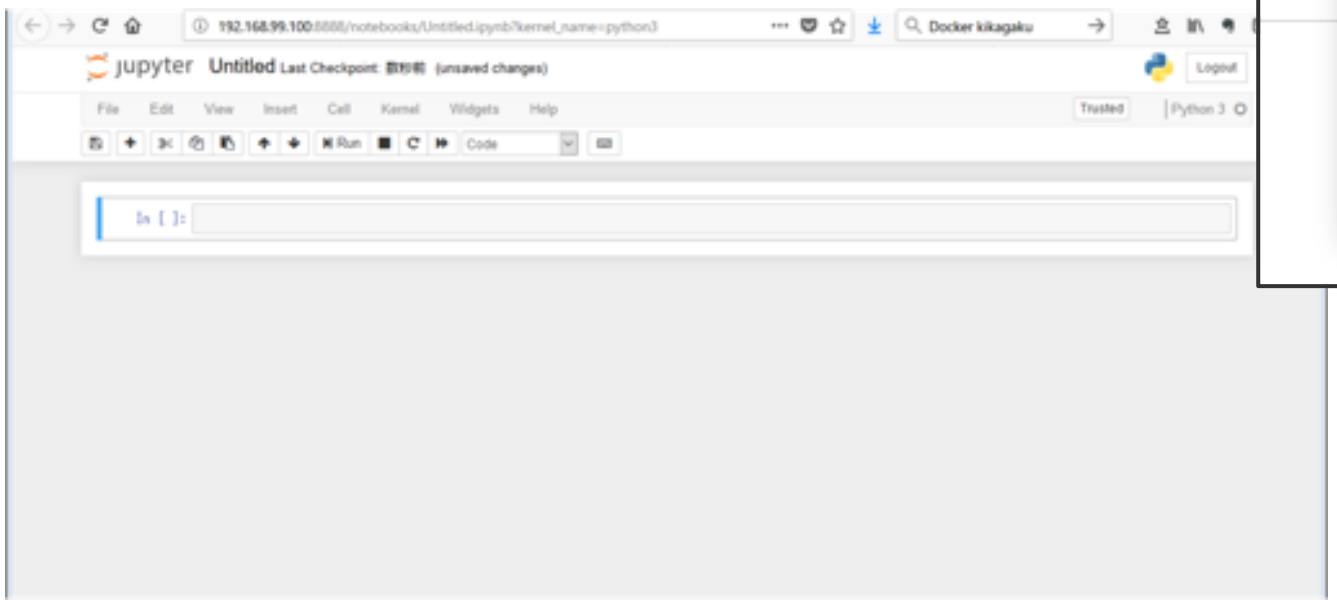
[Enter] で改行

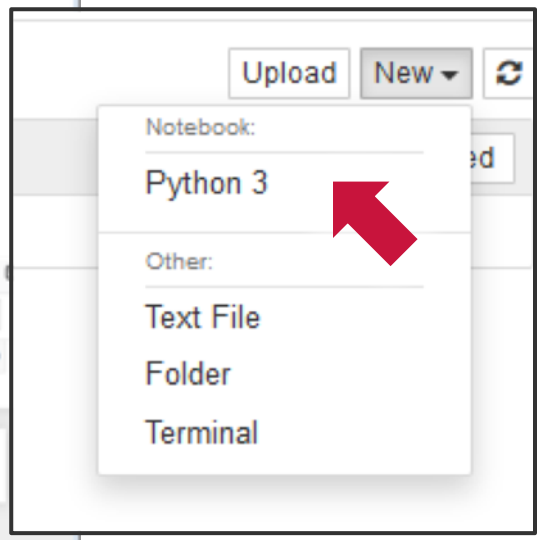
[Shift] + [Enter] で実行

[Alt/Option] + [Enter]でセル追加

Jupyter Notebook

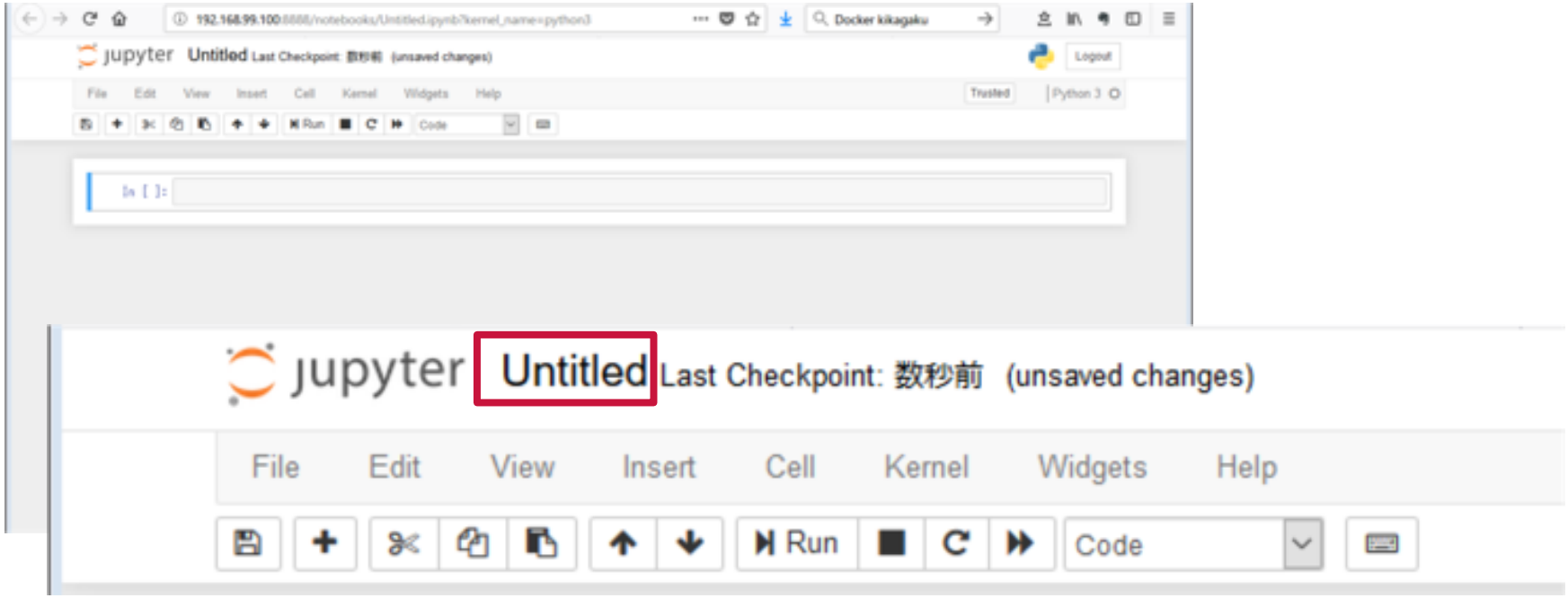






New → Python3 を選択
→新しいウィンドウが開く
(Notebook)

Jupyter Notebook



Notebookの名前を Untitled → HelloJupyter にリネームしてみよう

Rename
または [Enter] で変更

Jupyter Notebook

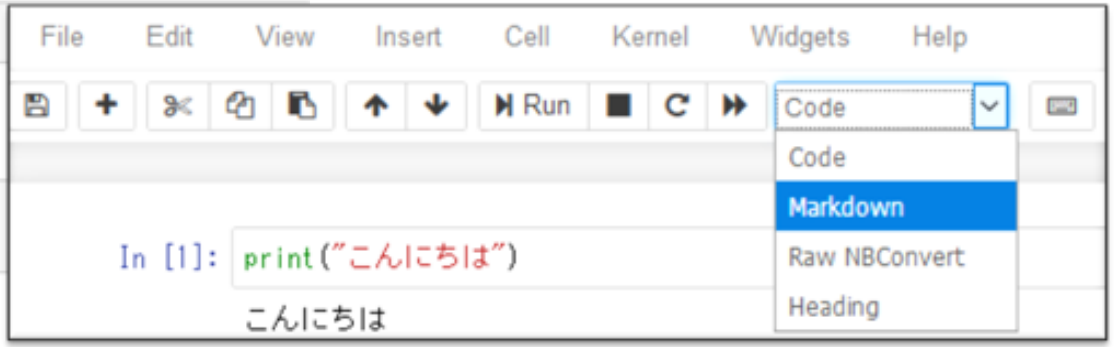

```
In [1]: print("こんにちは")
        こんにちは

In [2]: a,b = 2, 3

In [3]: a + b
Out[3]: 5

        オブジェクトの中身を確認

In [4]: b = dict(iscie=5, sice=63)

In [5]: b?

        オブジェクト名の検索

In [6]: import scipy as sp
        sp.*lu*?

        履歴の利用

In [7]: %history -no 2-3
           2: a,b = 2, 3
           3: a + b
        5
```

← コードセル

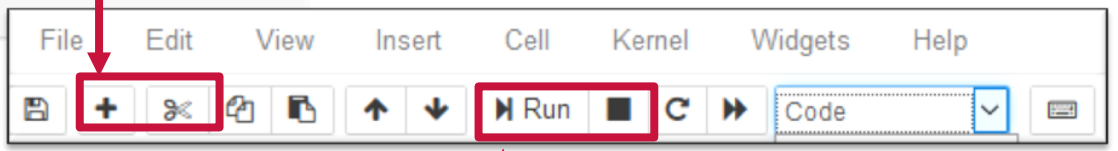
← コードセルの実行結果

← Markdown

← Code



コードセルの追加・削除



コードセルの実行・停止

誰でも綺麗なコードが書ける


```
# Setup
n = 0
# Loop
for n in range(10):
    n = n + 1
    # The % is the modulo operator
    if n % 2 == 0:
        print(n)
```

ブロックをインデントで表現
インデント = 半角スペース4つ!

#「#」をつけるとコメントアウト
#複数行の場合は，’’’　文１　文２　’’’ のように「’’’」で囲む

数値と四則演算

・数値は整数，実数，複素数のほかに，２進数や16進数も扱える

整数　1, 2, 3, -3, -2, -1

実数　1.5, 3.1415

複素数　1+3j, 2+1j, 3.2+5.6j

```
0b1010 # 2進数 → 10
0xA1 # 16進数 → 161
```

・数値の四則計算やべき乗の計算ができる

```
加算　1 + 2    #(=3)
減算　5 - 3    #(=2)
乗算　4 * 6    #(=24)
除算　10/ 5    #(=2.0)
```

```
商　　17 // 5   # (=3)
余　　17 % 5    # (=2)
べき乗　3 ** 2  # (=3*3=9)
```

変数と型

- **変数の定義は，「＝」を使う**
- **型宣言は必要ない**

int型　　x = 1

float型　y = 1.0

complex型　z = 1+3j

bool型　ok = True

　　ng = False

str型　　msg = ‘Control’　文字列はシングルクオーテションで囲む

- **変数名の決まり**
  - 名前の最初に数字は使えない
  - if や for, lambda などの予約語は使えない
  - 大文字と小文字は区別される

リスト

・**リストの定義と参照方法**　インデックスは０から始まる

```
data1 = [3, 5, 2, 4, 6, 1]
```

0　1　2　3　4　5　：先頭から参照するとき

-6　-5　-4　-3　-2　-1　：末尾から参照するとき

`data1[0]` → 3　　`data1[-2]` → 6

・**スライス**　開始インデックス～終了インデックス－１

`data1[2:4]` → 2, 4　　`data1[0:2]` → 3, 5

・**二次元のリスト**　入れ子にする　　`data1[0][1]` → 5

```
data2 =  [ [3, 5, 2], [4, 6, 1] ]
```

リスト

・メソッド

```
L = [1, 2, 3, 4]

L.append(5)
del L[1]
print(L)
>>> [1, 3, 4, 5]

L.extend([6, 7])
print(L)
>>> [1, 3, 4, 5, 6, 7]
```

要素の追加

要素の削除

del L　とするとリストごと削除される
L.pop(1) の方が安全

リストの連結

L = L+[6,7] でもOK

リスト

・浅いコピーと深いコピー

```
x = [ 1, 2, 3]
y = x
y[0] = 10
print(y) # [10, 2, 3]と出力される
print(x) # xの値も[10, 2, 3]に変更される
```

```
[10, 2, 3]
[10, 2, 3]
```

```
x = [ 1, 2, 3]
y = x.copy()
y[0] = 10
print(y) # [10, 2, 3]と出力される
print(x) # xの値は変更されずに[1, 2, 3]と出力される
```

```
[10, 2, 3]
[1, 2, 3]
```

copy()を使う

“＝”だとアドレスがコピー

```
id(x)
```

```
120856690432
```

```
id(y)
```

```
120856690432
```

```
id(x)
```

```
120857132640
```

```
id(y)
```

```
120855504048
```

タプル

・タプルの定義

```
tuple = (1,2,3,4)
```
丸括弧（）で囲む

```
tuple[0] → 1
```

リストのようものだが，書き換えができない = 読み込み専用の変数

```
tuple[0]=5
```
実行できない

```
data = [1,2,3,4] #リスト
data[0] = 5
```
実行できる → [5,2,3,4] となる

for分（ループ）


```
for 変数 in オブジェクト:
    実行する処理1
    実行する処理2
```

インデント（空白４個）

最後に「：」をつける

```
for i in [0, 1, 2]:
    print(i)
>>> 0, 1, 2
```

配列の各要素をiに代入し
繰り返し処理

```
for i in range(0,3):
    print(i)
>>> 0, 1, 2
```

0から1ずつ増える3より
小さい数のリストを生成

range(3)でもよい

if文（条件分岐）


```
if 条件式:
    条件成立で実行する処理
else:
    条件不成立で実行する処理
```

インデント（空白４個）

比較演算

==	等しい
!=	等しくない
>	より大きい
>=	以上
<	未満
<=	以下

```
for i in range(5):
    if i < 2:
        print('%d < 2' % i)
    elif i==2:
        print('%d == 2' % i)
    else:
        print('2 < %d < 5' % i)
```

i < 2の場合

i = 2の場合

それ以外の場合

```
>>> 0 < 2
>>> 1 < 2
>>> 2 == 2
>>> 2 < 3 < 5
>>> 2 < 4 < 5
```

ライブラリ

- NumPy：数値計算の基本ライブラリ
- Matplotlib：2次元，３次元プロットのライブラリ
- Sympy：数式処理のライブラリ
- Python-Control：制御工学のライブラリ

（SciPy）：数値計算アルゴリズム（信号処理，最適化，統計）

（Pandas）：データ分析

（Scikit-learn）：機械学習のライブラリ

ライブラリの組み込み

import または，from でライブラリを組み込む

```
import ライブラリ名
import ライブラリ名 as 別名
```

```
import numpy as np
x = np.arange(0, 10, 0.1)
```

np で組み込んで
np.関数名で実行

```
from ライブラリ名  import 関数名
```

```
from scipy.integrate import odeint
x = odeint(system, x0, t)
```

関数名だけで実行できる

Numpy

Python：Numpyライブラリ

```
import numpy as np
```

等間隔の配列の作成

arange(start, stop, [step])

```
T = np.arange(0, 10, 1)
>>> [0 1 2 3 4 5 6 7 8 9]
```

行列・ベクトルの作成

array([[1行目], [2行目] , [3行目]])

data[:, 1] >>> [1列目]

三角関数　sin, cos　　**円周率**　pi

```
import numpy as np
```

```
A = np.array([ [1, 2], [-3, 4]])
print(A)
```

```
[[ 1  2]
 [-3  4]]
```

転置　np.transpose(行列)
行列.T

```
print(A.T)
```

```
[[ 1 -3]
 [ 2  4]]
```

```
B = np.abs(A)
print(B)
```

```
[[1 2]
 [3 4]]
```

```
np.linalg.det(A)
```

行列式

```
10.000000000000002
```

```
np.linalg.matrix_rank(A)
```

```
2
```

ランク

```
x = np.array([1, 2])
print(x)
```

```
[1 2]
```

```
np.linalg.norm(x)
```

ノルム

```
2.23606797749979
```

```
w, v = np.linalg.eig(A)
print('eigenvalue=',w)
print('eigenvector=\n',v)
```

固有値

```
eigenvalue= [2.5+1.93649167j 2.5-1.93649167j]
eigenvector=
 [[0.38729833-0.5j 0.38729833+0.5j]
 [0.77459667+0.j  0.77459667-0.j ]]
```

```
C = np.linalg.inv(A)
print(C)
```

逆行列

```
[[ 0.4 -0.2]
 [ 0.3  0.1]]
```

Numpy


```
data3 = np.arange(0, 5, 0.1)
```

```
data3
```

```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
       1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5,
       2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8,
       3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9])
```

```
data3[0::2]
```

```
array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4,
       2.6, 2.8, 3. , 3.2, 3.4, 3.6, 3.8, 4. , 4.2, 4.4, 4.6, 4.8])
```

偶数番号（０から１個とばし）

```
data3[1::2]
```

```
array([0.1, 0.3, 0.5, 0.7, 0.9, 1.1, 1.3, 1.5, 1.7, 1.9, 2.1, 2.3, 2.5,
       2.7, 2.9, 3.1, 3.3, 3.5, 3.7, 3.9, 4.1, 4.3, 4.5, 4.7, 4.9])
```

奇数番号（１から１個とばし）

```
data3[::-1]
```

```
array([4.9, 4.8, 4.7, 4.6, 4.5, 4.4, 4.3, 4.2, 4.1, 4. , 3.9, 3.8, 3.7,
       3.6, 3.5, 3.4, 3.3, 3.2, 3.1, 3. , 2.9, 2.8, 2.7, 2.6, 2.5, 2.4,
       2.3, 2.2, 2.1, 2. , 1.9, 1.8, 1.7, 1.6, 1.5, 1.4, 1.3, 1.2, 1.1,
       1. , 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0. ])
```

反転

```
data3[1:15:2]
```

```
array([0.1, 0.3, 0.5, 0.7, 0.9, 1.1, 1.3])
```

15番目まで２個飛ばし

Matplotlib

Python：matplotlibライブラリ

```
import matplotlib.pyplot as plt
```

plot(横軸データ，縦軸データ，[オプション])

lw，太さ

ls，種類

記号	線種	記号	線種
-	実線	--	破線
-.	一点鎖線	:	点線

marker，種類

color，色

文字	色	文字	色
b	青	g	緑
r	赤	c	シアン
m	マゼンタ	y	黄
k	黒	w	白

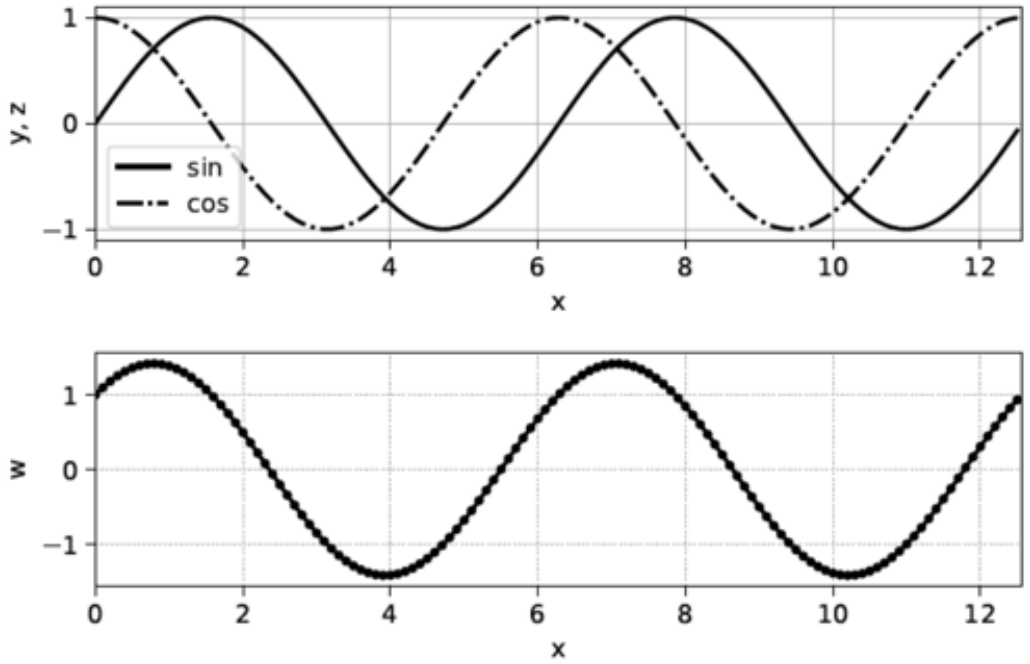
```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 4 * np.pi, 0.1)
y = np.sin(x)

plt.plot(x, y) # 横軸x，縦軸yでプロット
plt.xlabel('x') # x 軸のラベルを設定
plt.ylabel('y') # y 軸のラベルを設定
plt.grid() # グリッドの表示
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 4 * np.pi, 0.1)
y = np.sin(x)

fig, ax = plt.subplots() # FigureとAxesオブジェクトの作成
ax.plot(x, y) # Axesオブジェクトの中にグラフを作成
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.grid()
plt.show()
```

おススメ！

Python演習１：Matplotlib


```
fig, ax = plt.subplots(2,1)
x = np.arange(0, 4 * np.pi, 0.1)
y = np.sin(x)
z = np.cos(x)
w = y + z

ax[0].plot(x, y, ls='-', label='sin', color='k')
ax[0].plot(x, z, ls='-.', label='cos', color='k')
ax[0].set_xlabel('x')
ax[0].set_ylabel('y, z')
ax[0].set_xlim(0, 4*np.pi)
ax[0].grid()
ax[0].legend() #凡例
ax[1].plot(x, w, color='k', marker='.')
ax[1].set_xlabel('x')
ax[1].set_ylabel('w')
ax[1].set_xlim(0, 4*np.pi)
ax[1].grid(ls=':’)

fig.tight_layout()
```



Python Control

関数名	使用例	説明
tf	sys = tf(num, den)	伝達関数モデルの定義
ss	sys = ss(A, B, C, D)	状態空間モデルの定義
tfdata	[[num]], [[den]] = tfdata(sys)	伝達関数の分子・分母多項式の抽出
ssdata	A, B, C, D = ssdata(sys)	状態空間の $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ の抽出
tf2ss	sysss = tf2ss(systf)	伝達関数から状態空間への変換
ss2tf	systf = ss2tf(sysss)	状態空間から伝達関数への変換
series	sys = series(sys1, sys2)	システムの直列結合
parallel	sys = parallel(sys1, sys2)	システムの並列結合
feedback	sys = feedback(sys1, sys2, sign=-1)	システムのフィードバック結合
minreal	sysmin = minreal(sys)	最小実現

関数名	使用例	説明
pade	sys = pade(h, n)	無駄時間要素のパデ近似 (h: 無駄時間, n: 次数)
acker	F = -acker(A, B, p)	1 入力系に対する極配置法
place	F = -place(A, B, p)	多入力系に対する極配置法
lqr	F, X, E = lqr(A, B, Q, R)	最適レギュレータ $J = \int_0^\infty (\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^\top \boldsymbol{R}\boldsymbol{u})\mathrm{d}t$ を最小化 F:ゲイン, X:リカッチ方程式の解, E:閉ループ極
care	X, E, F = care(A, B, Q, R)	リカッチ方程式の解 $\boldsymbol{A}^\top \boldsymbol{P} + \boldsymbol{P}\boldsymbol{A} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top \boldsymbol{P} + \boldsymbol{Q} = 0$ X: 解, F:$\boldsymbol{F} = -\boldsymbol{R}^{-1}\boldsymbol{B}^\top \boldsymbol{P}$, E: 閉ループ極
c2d	Pd = c2d(Pc, method='zoh') Pd = c2d(Pc, method='tustin')	連続時間モデルから離散時間モデルへの変換 zoh: 0 次ホールド, tustin: 双一次変換

関数名	使用例	説明
pole	p = pole(sys)	伝達関数の極を求める
zero	z = zero(sys)	伝達関数の零点を求める
dcgain	k = dcgain(sys)	直流ゲインを求める
step	y, t = step(sys, Td)	ステップ応答
impulse	y, t = impulse(sys, Td, X0)	インパルス応答
initial	y, t = initial(sys, Td, X0)	初期値応答（零入力応答）
lsim	y, t, x0 = lsim(sys, Ud, Td, X0)	任意の入力に対する時間応答
bode	gain, phase, w = bode(sys, W)	ボード線図
nyquist	re, im, w = nyquist(sys, W)	ナイキスト線図
margin	gm, pm, wpc, wgc = margin(sys)	ゲイン余裕, 位相余裕 位相交差周波数, ゲイン交差周波数の計算
freqresp	g, ph, w = freqresp(sys, Wc)	特定の周波数におけるゲインと位相の計算
ctrb	Uc = ctrb(A, B)	可制御性行列の計算
obsv	Uo = obsv(A, C)	可観測性行列の計算

Python: controlライブラリ

```
from control.matlab import *
```

Python演習 1

1から50までの和を計算して表示

```
#普通の方法
s = 0
for x in range(1,51):
     s += x
print(s)

#sumを使う
print(sum(range(1,51)))

#generator式を使う
print(sum(x for x in range(1,51)))

#reduceを使う
import functools
print(functools.reduce(lambda x,y: x+y, range(1,51)))
```

reduce関数（高階関数）
→ 配列内の要素を順次足す

無名関数

Python演習１：数式処理

数式処理で方程式の解を求める

```
import sympy as sp
sp.init_printing()

s = sp.Symbol('s')

p1 = sp.solve(2 * s**2 +5*s+3, s)
```

$$2s^2 + 5s + 3 = 0$$

```
p2 = sp.solve(s**3 +s**2+s+1, s)
```

$$s^3 + s^2 + s + 1 = 0$$

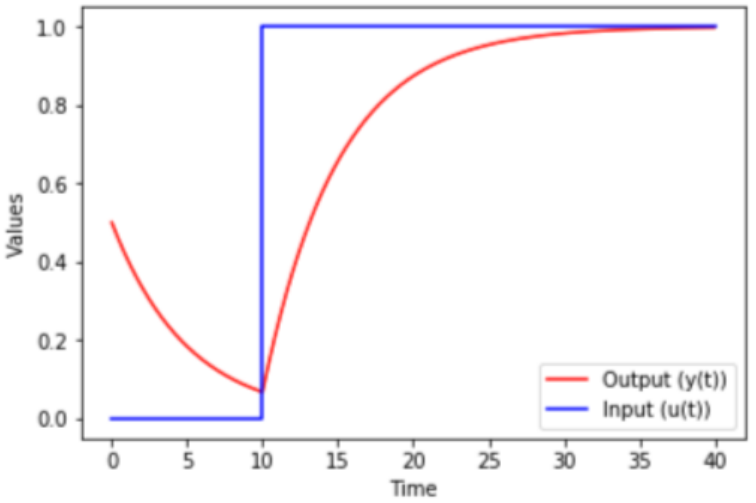
```
p1, p2
```

MapleやMathematica, Maxima と同様のことができます
たとえば，ラプラス変換や逆ラプラス変換も可能です
各自調べてみましょう

Python演習１：微分方程式を解く


```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def system(y, t):
    if t < 10.0:
        u = 0
    else:
        u = 1
    dydt = (-y + u)/5.0
    return dydt

y0 = 0.5
t = np.arange(0, 40, 0.04)
y = odeint(system, y0, t)

fig, ax = plt.subplots() # FigureとAxesオブジェクトの作成
ax.plot(t, y, 'r-', label='Output (y(t))')
ax.plot([0, 10, 10, 40], [0, 0, 1, 1], 'b-', label='Input (u(t))')
ax.set_xlabel('Time')
ax.set_ylabel('Values')
ax.grid()
ax.legend(loc='best’)
```

余力のある人は，u(t) = sin(t) の結果も見てみましょう



勉強会の概要

制御工学の基礎
- 制御とは，フィードバック制御，制御系設計

Pythonプログラミングの基礎
- Jupyter Notebook の使い方
- Pythonプログラミング　★ Python演習 1

制御系設計のためのモデル
- 伝達関数モデル，ブロック線図，時間応答，周波数応答　★ Python演習 2

Part 2

制御系設計のためのモデル（復習）
- 伝達関数モデル，ブロック線図，時間応答，周波数応答

伝達関数モデルを用いた制御系設計
- 閉ループ系の設計仕様，PID制御，モデルマッチング　★ Python演習 3

Part 3

ループ整形による制御系設計
- 開ループ系の設計仕様，位相進み・遅れ補償　★ Python演習 4

状態空間モデルを用いた制御系設計
- 状態空間モデル，状態フィードバック　★ Python演習 5

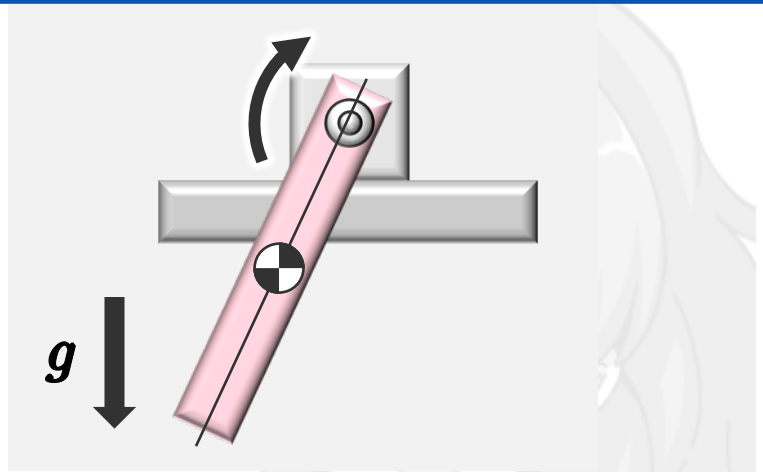
姉妹のよくある日常会話

姉：希波（きなみ）　２３歳

妹：望結（みゆう）　１８歳

姉妹のよくある日常会話

ねぇねぇ，このアームをいい感じに
動かしたいんだけど・・・



そうだね・・・数学モデルね．
運動方程式はこんな感じよ

制御器を作りたいんだね
じゃぁ，お菓子とモデル頂戴

試行錯誤で制御系設計は辛いから
モデルを使って設計するの

このままじゃ使えないよ～せめて，
伝達関数とか状態方程式にしといてよ

★設計用モデルが必要．状況に合わせて使い分ける

動的システムと静的システム

静的システム

現在の入力で
現在の出力が決まる

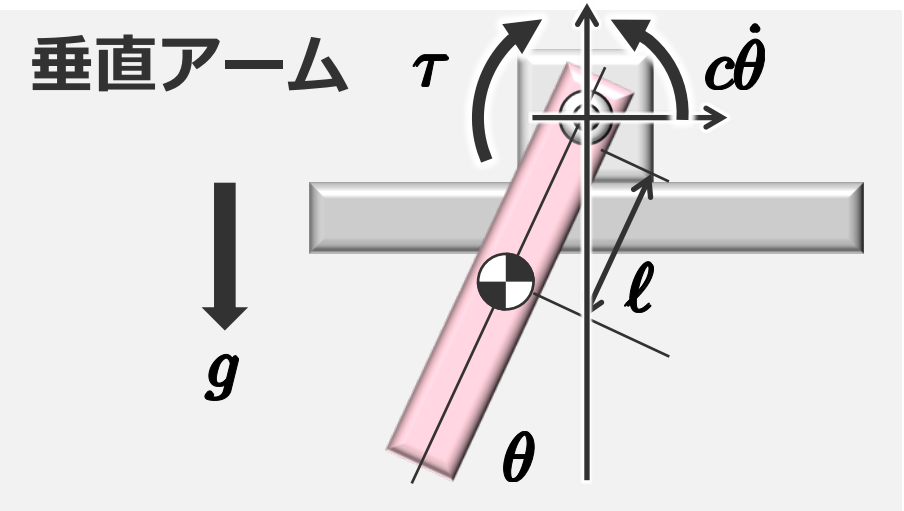
→ **メモリ不要**

動的システム

過去から現在までの入力で
現在の出力が決まる

→ **メモリ必要 ＝ 微分方程式**

機械系のモデル



運動方程式

$$J\ddot{\theta}(t) = -c\dot{\theta}(t) - mg\ell\sin\theta(t) + \tau(t)$$

⬇ 線形化

$$J\ddot{\theta}(t) = -c\dot{\theta}(t) - mg\ell\theta(t) + \tau(t)$$

⬇ $u(t) = \tau(t) \quad y(t) = \theta(t)$

$$J\ddot{y}(t) + c\dot{y}(t) + mg\ell y(t) = u(t)$$

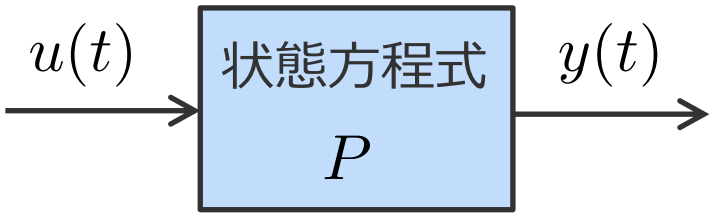
⬇ $x(t) = \begin{bmatrix} \theta(t) \\ \dot{\theta}(t) \end{bmatrix}$ **状態**（選び方に自由度あり）

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ -\frac{mg\ell}{J} & -\frac{c}{J} \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \frac{1}{J} \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(t)$$

状態方程式

線形動的システム



設計モデルの関係

高階の常微分方程式

$$\frac{\mathrm{d}^n}{\mathrm{d}t^n}y(t)+a_{n-1}\frac{\mathrm{d}^{n-1}}{\mathrm{d}t^{n-1}}y(t)+\cdots+a_1\frac{\mathrm{d}}{\mathrm{d}t}y(t)+a_0y(t)$$
$$=b_m\frac{\mathrm{d}^m}{\mathrm{d}t^m}u(t)+b_{m-1}\frac{\mathrm{d}^{m-1}}{\mathrm{d}t^{m-1}}u(t)+\cdots+b_1\frac{\mathrm{d}}{\mathrm{d}t}u(t)+b_0u(t)$$

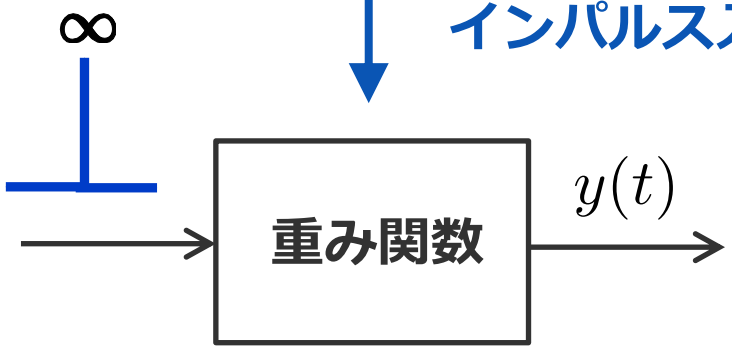
行列表現

状態空間モデル

$$P:\begin{cases}\dot{\boldsymbol{x}}(t) &= \boldsymbol{A}\boldsymbol{x}(t)+\boldsymbol{B}u(t)\\ y(t) &= \boldsymbol{C}\boldsymbol{x}(t)\end{cases}$$

解（応答）が求めやすい！

インパルス入力



重み関数
（インパルス応答）

$$y(t)=p(t)=\boldsymbol{C}e^{\boldsymbol{A}t}\boldsymbol{B}$$

→ **たたみこみ積分**
（任意入力の応答）

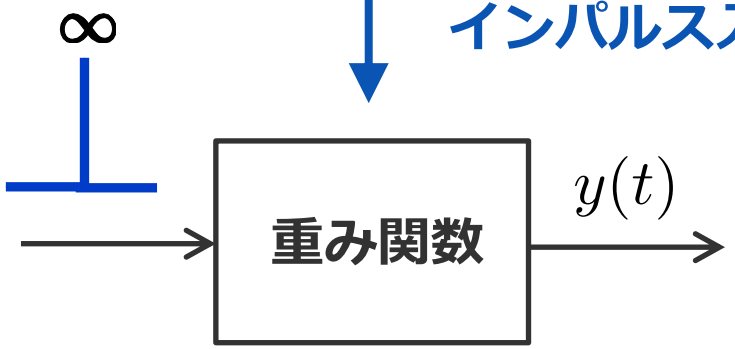
$$y(t)=\int_0^t p(t-\tau)u(\tau)\mathrm{d}\tau$$

時間領域

設計モデルの関係

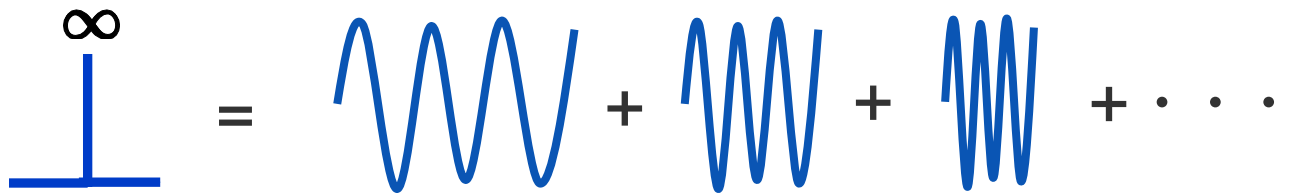
行列表現

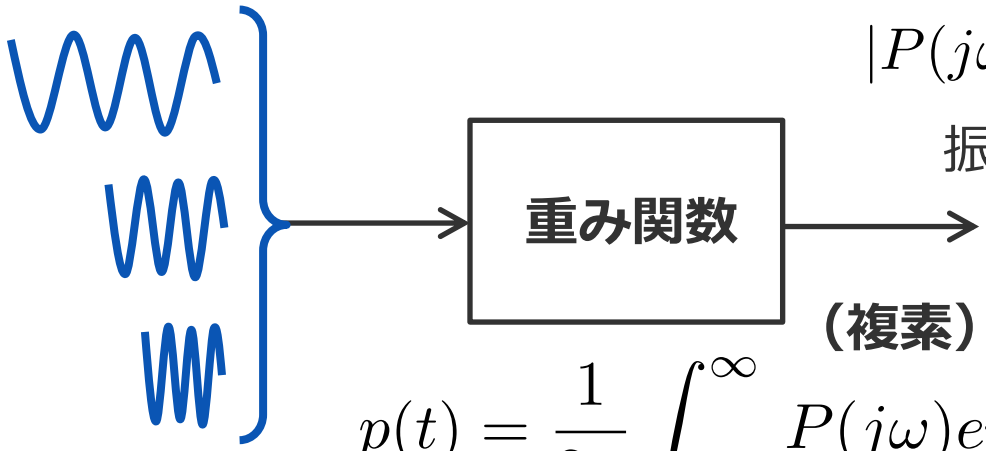
インパルス入力



時間領域

現実世界では，インパルス入力は使いにくい・・・
「インパルス入力＝正弦波の重ね合わせ」だと考える！





$|P(j\omega)|$ $\angle P(j\omega)$

振幅と位相が変化

（複素）フーリエ級数展開！

$$p(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} P(j\omega)e^{j\omega t}\mathrm{d}\omega$$

設計モデルの関係

$u(t)$ → **微分方程式** → $y(t)$

周波数領域

システムが安定（有界入力有界出力）として，定常状態での $y(t)$ の

振幅 と **位相** に注目

行列表現

$u(t)$ → **状態方程式** → $y(t)$

正弦波入力 → 周波数伝達関数 → $y(t)$

$|P(j\omega)|$

$\angle P(j\omega)$

インパルス入力

∞ → **重み関数** → $y(t)$

$$P(j\omega) = \int_0^\infty p(t)e^{-j\omega t}\mathrm{d}t = \boldsymbol{C}(j\omega I - \boldsymbol{A})^{-1}\boldsymbol{B}$$

フーリエ変換

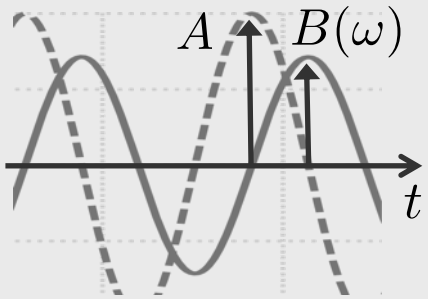
時間領域

$$y(t) = p(t) = \boldsymbol{C}e^{\boldsymbol{A}t}\boldsymbol{B}$$

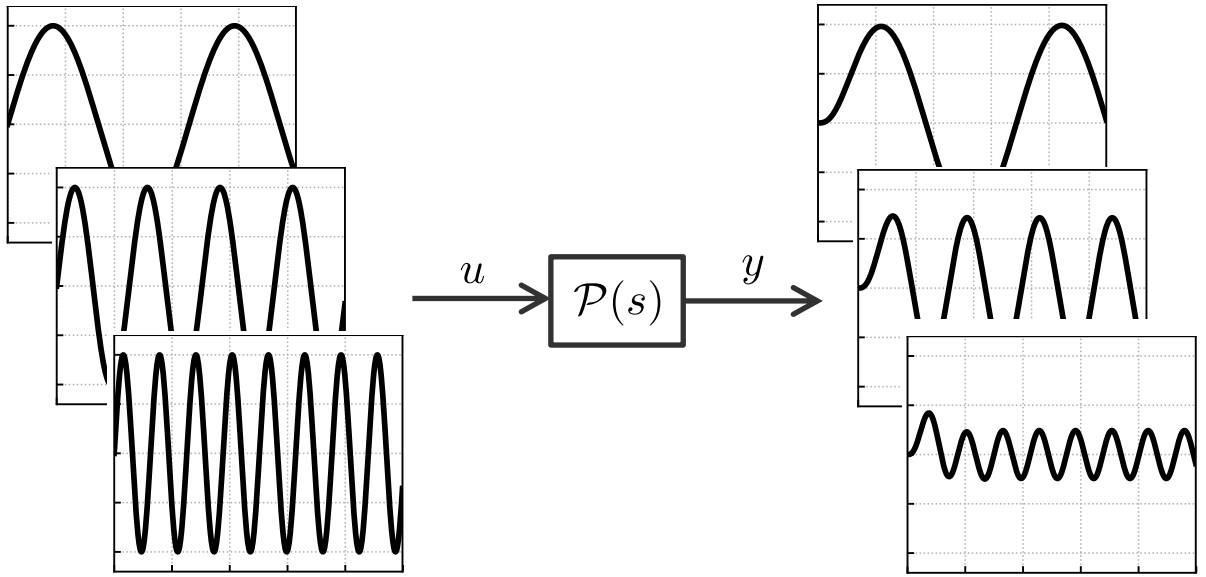
- **ボード線図（振幅，位相）**
- **ナイキスト線図**

周波数特性

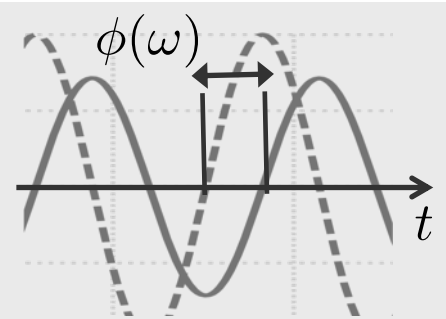
振幅が変わる



$$|\mathcal{P}(j\omega)| = \frac{B(\omega)}{A}$$



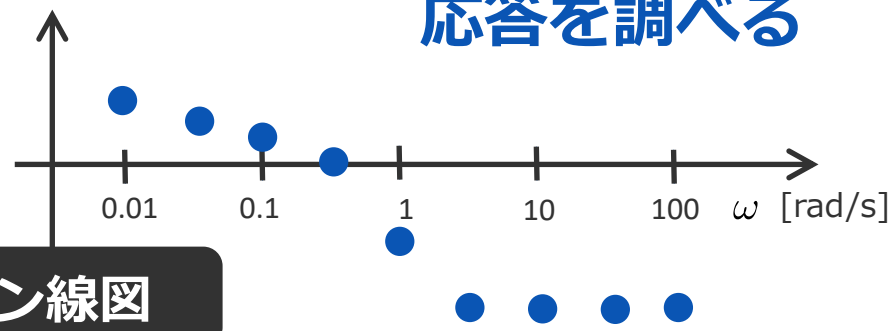
位相がズレる



$$\angle\mathcal{P}(j\omega) = \phi(\omega)$$

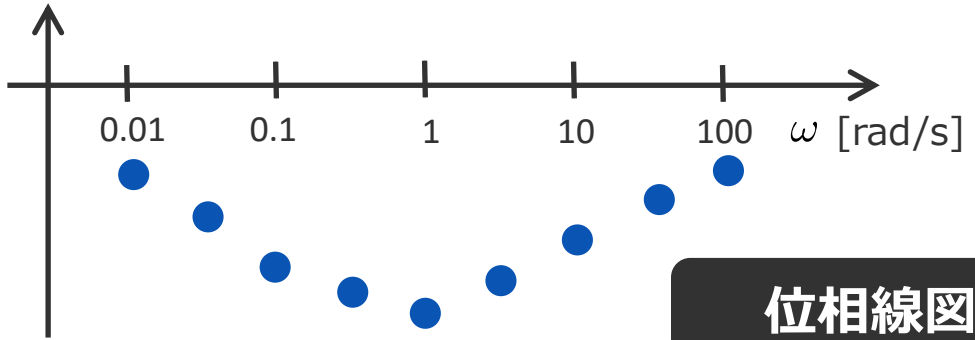
周波数を変化させて応答を調べる

振幅比



ゲイン線図

位相

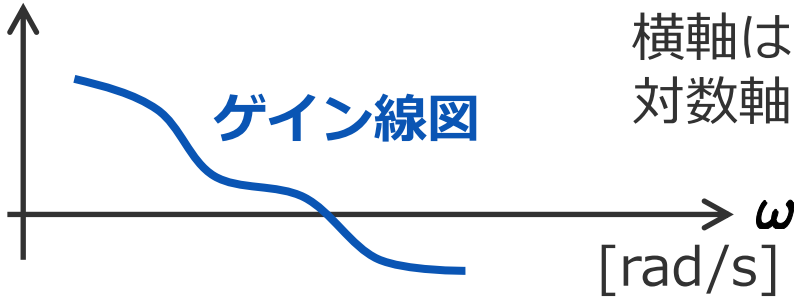


位相線図

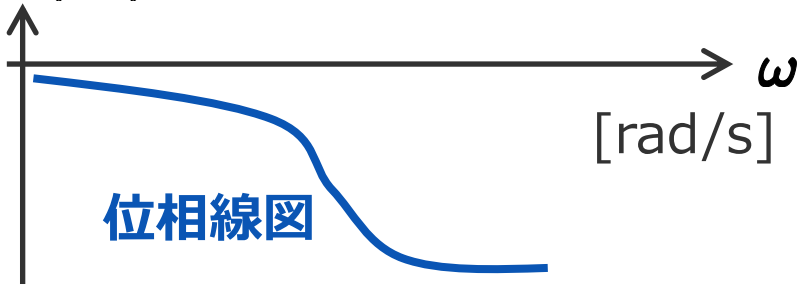
周波数特性

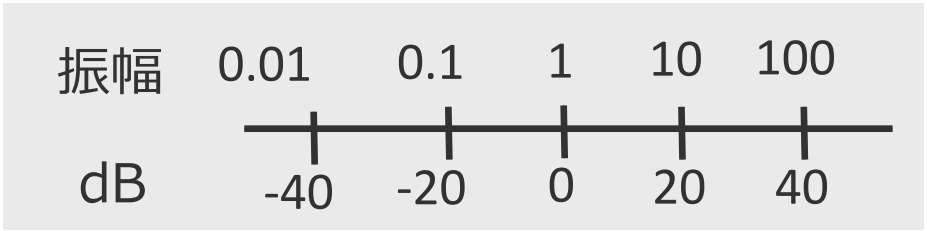
ボード線図

$20\log_{10}|P(j\omega)|$ [dB]

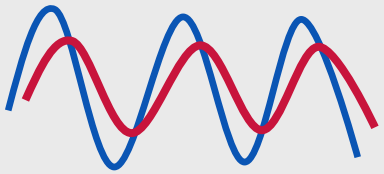


$\angle P(j\omega)$ [deg]





周波数応答



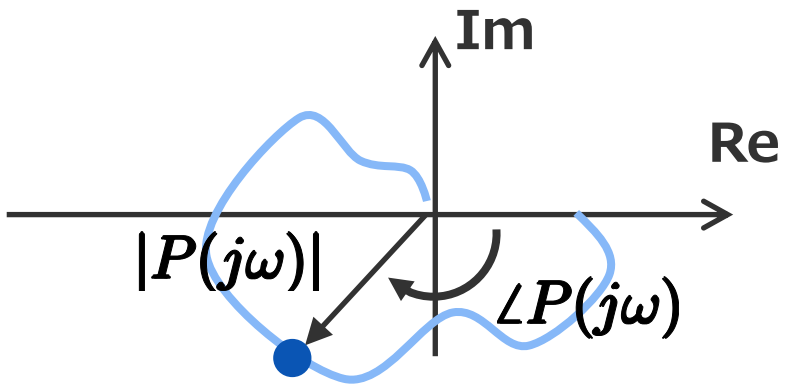
$$u(t) = \sin\omega t$$

$$y(t) = |P(j\omega)|\sin(\omega t + \phi)$$

$$\phi = \angle P(j\omega)$$

ナイキスト線図 $\omega = -\infty \to \infty$

$$P(j\omega) = \alpha(\omega) + j\beta(\omega)$$



設計モデルの関係

$u(t)$ → **微分方程式** → $y(t)$

周波数領域

システムが安定（有界入力有界出力）として，定常状態での $y(t)$ の

振幅 と **位相** に注目

行列表現

$u(t)$ → **状態方程式** → $y(t)$

正弦波入力 → **周波数伝達関数** → $y(t)$

$$|P(j\omega)|$$
$$\angle P(j\omega)$$

インパルス入力

∞ → **重み関数** → $y(t)$

$$P(j\omega) = \int_0^\infty p(t)e^{-j\omega t}\mathrm{d}t = \boldsymbol{C}(j\omega I - \boldsymbol{A})^{-1}\boldsymbol{B}$$

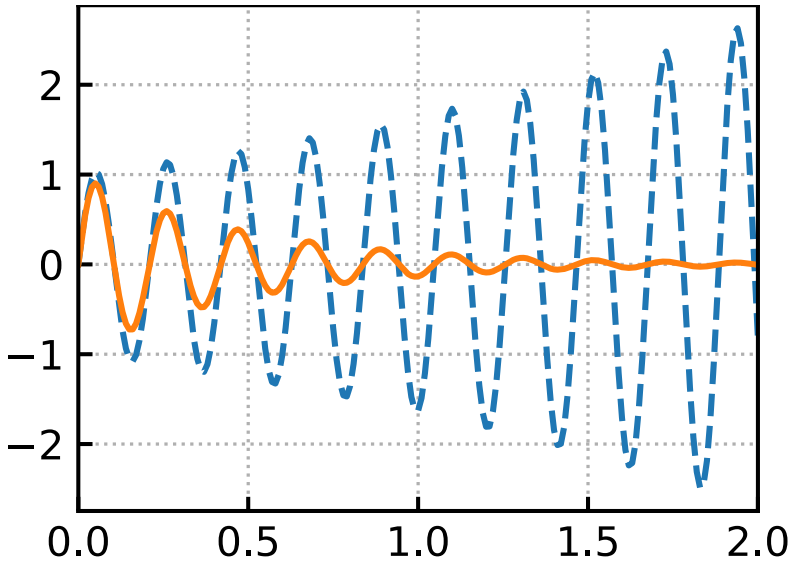
フーリエ変換

時間領域

$$y(t) = p(t) = \boldsymbol{C}e^{\boldsymbol{A}t}\boldsymbol{B}$$

- **ボード線図（振幅，位相）**
- **ナイキスト線図**

設計モデルの関係

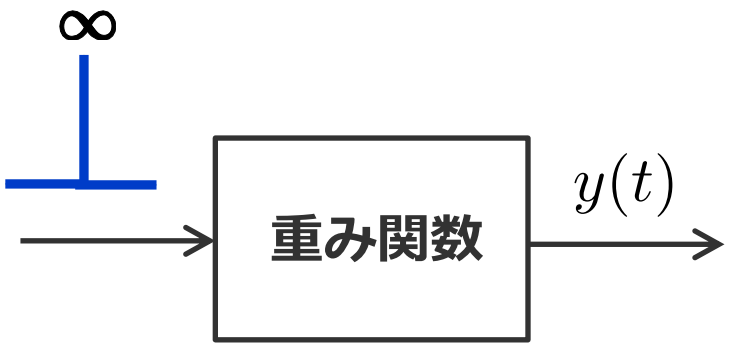


フーリエ変換を計算するには，
システムが安定でないとダメ

重み関数 $p(t)$ に適当な指数関数をかけて“圧縮する”と安定になる！

$$P_L(j\omega) = \int_0^\infty p(t)e^{-\sigma t}e^{-j\omega t}\mathrm{d}t$$

$s = \sigma + j\omega$ ↓ フーリエ変換の拡張



$$y(t) = p(t) = \boldsymbol{C}e^{\boldsymbol{A}t}\boldsymbol{B}$$

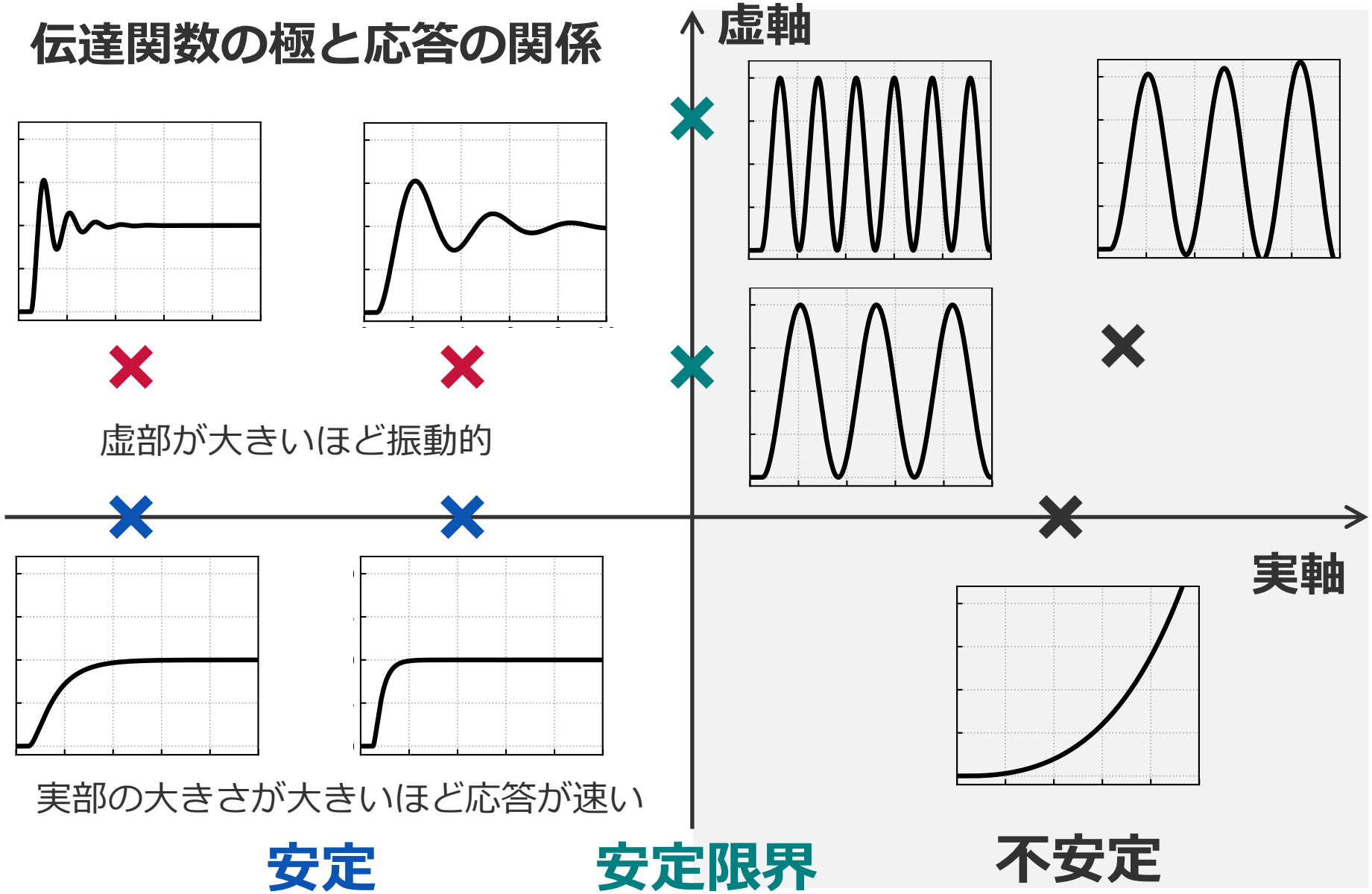
ラプラス変換

伝達関数

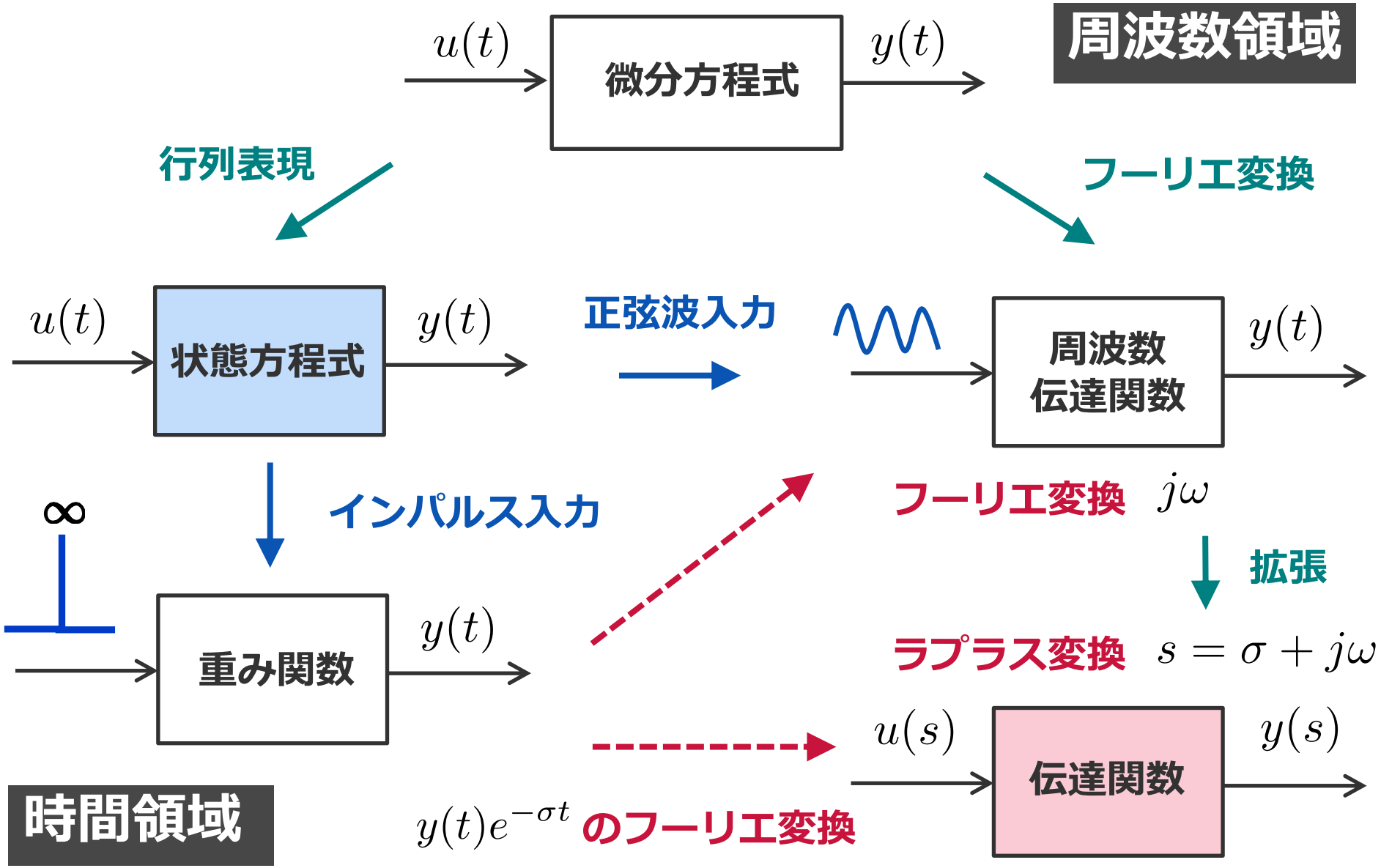
$$P(s) := P_L(j\omega) = \int_0^\infty p(t)e^{-st}\mathrm{d}t$$
$$= \boldsymbol{C}(sI - \boldsymbol{A})^{-1}\boldsymbol{B}$$

極と振る舞いの関係

伝達関数の極と応答の関係



設計モデルの関係


$u(t)$
微分方程式
$y(t)$
周波数領域
行列表現
フーリエ変換
$u(t)$
状態方程式
$y(t)$
正弦波入力
周波数
伝達関数
$y(t)$
∞
インパルス入力
フーリエ変換 $j\omega$
拡張
重み関数
$y(t)$
ラプラス変換 $s = \sigma + j\omega$
$u(s)$
伝達関数
$y(s)$
時間領域
$y(t)e^{-\sigma t}$ のフーリエ変換

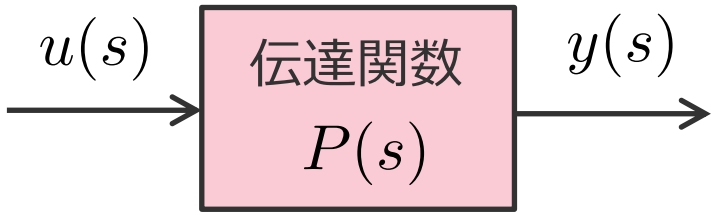
制御のためのモデル（設計モデル）

動的システム

$$\frac{\mathrm{d}^n}{\mathrm{d}t^n}y(t) + a_{n-1}\frac{\mathrm{d}^{n-1}}{\mathrm{d}t^{n-1}}y(t) + \cdots + a_1\frac{\mathrm{d}}{\mathrm{d}t}y(t) + a_0 y(t)$$
$$= b_m\frac{\mathrm{d}^m}{\mathrm{d}t^m}u(t) + b_{m-1}\frac{\mathrm{d}^{m-1}}{\mathrm{d}t^{m-1}}u(t) + \cdots + b_1\frac{\mathrm{d}}{\mathrm{d}t}u(t) + b_0 u(t)$$

伝達関数モデル　ラプラス変換を用いて代数方程式で表す

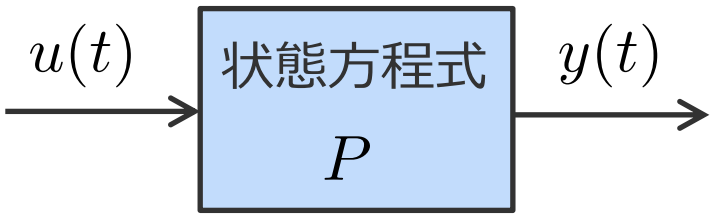
変換



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

分母多項式の根が伝達関数の極

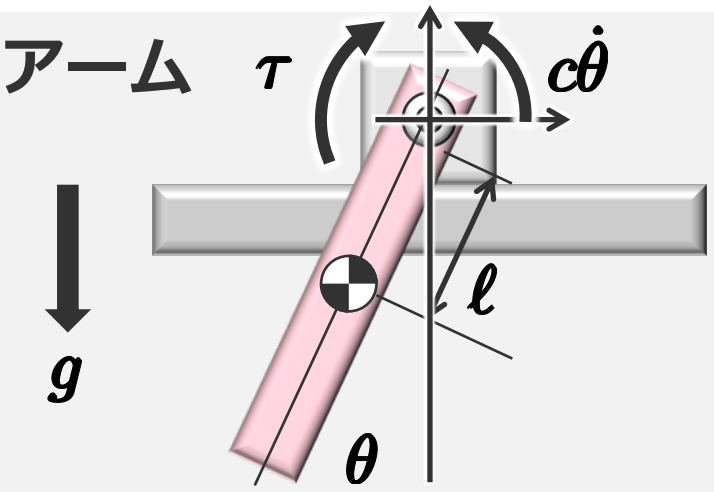
状態空間モデル　行列表現を用いて 1 階の微分方程式で表す



$$P : \begin{cases} \dot{\boldsymbol{x}}(t) &= \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}u(t) \\ y(t) &= \boldsymbol{C}\boldsymbol{x}(t) \end{cases}$$

機械系のモデル

垂直アーム



運動方程式

$$J\ddot{\theta}(t) = -c\dot{\theta}(t) - mg\ell \sin\theta(t) + \tau(t)$$

⬇ 線形化

$$J\ddot{\theta}(t) = -c\dot{\theta}(t) - mg\ell\theta(t) + \tau(t)$$

⬇ $y(t) = \theta(t) \quad u(t) = \tau(t)$

$$J\ddot{y}(t) + c\dot{y}(t) + mg\ell y(t) = u(t)$$

⬇ ラプラス変換

$$P(s) = \frac{y(s)}{u(s)} = \frac{1}{Js^2 + cs + mg\ell}$$

伝達関数

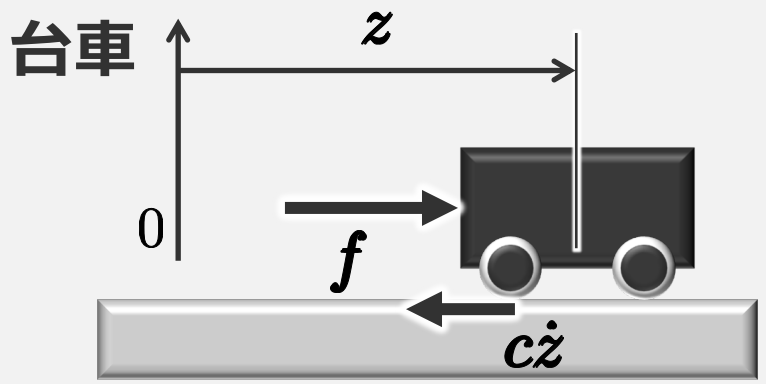
⬇ $x(t) = \begin{bmatrix} \theta(t) \\ \dot{\theta}(t) \end{bmatrix} \quad \begin{matrix} u(t) = \tau(t) \\ y(t) = \theta(t) \end{matrix}$

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ -\frac{mg\ell}{J} & -\frac{c}{J} \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \frac{1}{J} \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(t)$$

状態方程式

機械系のモデル



運動方程式

$$m\ddot{z}(t) = -c\dot{z}(t) + f(t)$$

状態の取り方には自由度がある

$$y(t) = \dot{z}(t) \quad u(t) = f(t)$$

$$m\dot{y}(t) + cy(t) = u(t)$$

ラプラス変換

$$P(s) = \frac{y(s)}{u(s)} = \frac{1}{ms + c}$$

伝達関数

$$y(t) = \dot{z}(t) \quad u(t) = f(t)$$

$$x(t) = \dot{z}(t) \qquad x(t) = \begin{bmatrix} z(t) \\ \dot{z}(t) \end{bmatrix}$$

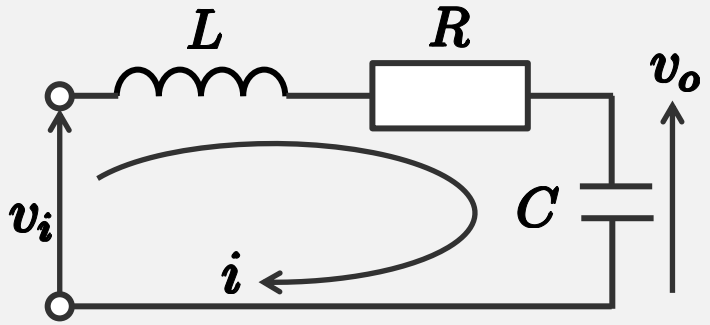
$$\dot{x}(t) = -\frac{c}{m}x(t) + \frac{1}{m}u(t) \qquad \dot{x}(t) = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{c}{m} \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u(t)$$

$$y(t) = x(t) \qquad y(t) = \begin{bmatrix} 0 & 1 \end{bmatrix} x(t)$$

状態方程式

電気系のモデル

RCL回路



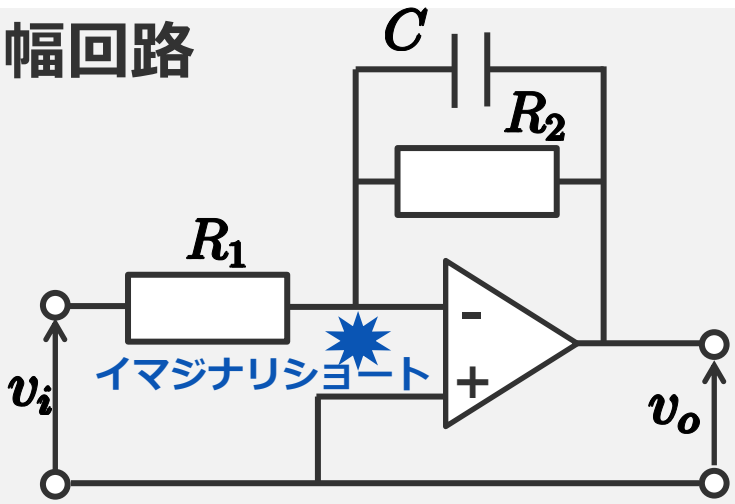
$$v_i = L\frac{\mathrm{d}i}{\mathrm{d}t} + Ri + \frac{1}{C}\int i\mathrm{d}t$$

$$v_o = \frac{1}{C}\int i\mathrm{d}t$$

$$y(t) = v_o(t) \quad u(t) = v_i(t)$$

$$P(s) = \frac{y(s)}{u(s)} = \frac{1}{CLs^2 + CRs + 1}$$

増幅回路



$$i_1 = \frac{v_i}{R_1} \quad i_2 = \frac{v_o}{R_2}$$

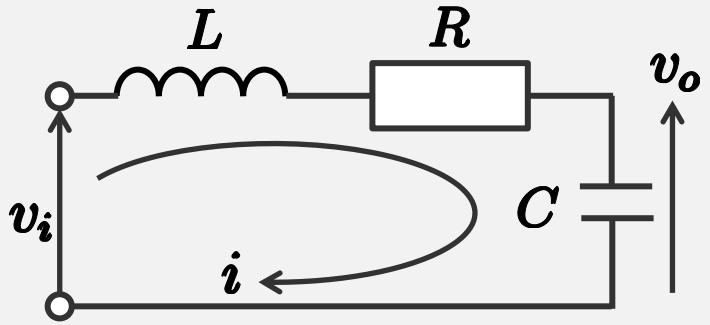
$$v_o = \frac{1}{C}\int i_3\mathrm{d}t \quad i_1 + i_2 + i_3 = 0$$

$$y(t) = v_o(t) \quad u(t) = v_i(t)$$

$$P(s) = \frac{y(s)}{u(s)} = -\frac{R_2}{R_1R_2Cs + R_1}$$

電気系のモデル

RCL回路

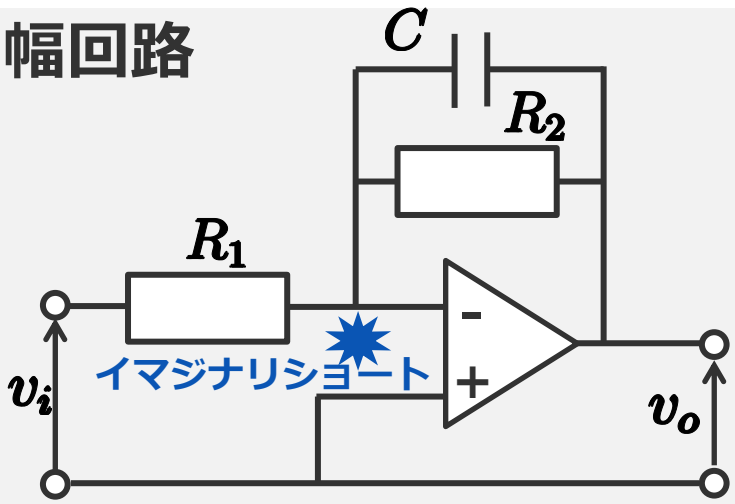


$$x(t) = \begin{bmatrix} \int i(\tau)\mathrm{d}\tau \\ i(t) \end{bmatrix} \qquad \begin{aligned} y(t) &= v_o(t) \\ u(t) &= v_i(t) \end{aligned}$$

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ -\frac{1}{LC} & -\frac{R}{L} \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} \frac{1}{C} & 0 \end{bmatrix} x(t)$$

増幅回路



$$x(t) = v_o(t) \qquad \begin{aligned} y(t) &= v_o(t) \\ u(t) &= v_i(t) \end{aligned}$$

$$\dot{x}(t) = -\frac{1}{CR_2}x(t) - \frac{1}{CR_1}u(t)$$

$$y(t) = x(t)$$

伝達関数モデルと状態空間モデルの関係

状態空間モデル

$$P:\begin{cases}\dot{\boldsymbol{x}}(t) &= \boldsymbol{A}\boldsymbol{x}(t)+\boldsymbol{B}u(t)\\ y(t) &= \boldsymbol{C}\boldsymbol{x}(t)\end{cases}$$

入力ー状態ー出力

一意に決まる

無数にある（状態の取り方次第）

伝達関数モデル

$$P(s)=\frac{y(s)}{u(s)}=\boldsymbol{C}(sI-\boldsymbol{A})^{-1}\boldsymbol{B}$$

入力ー出力

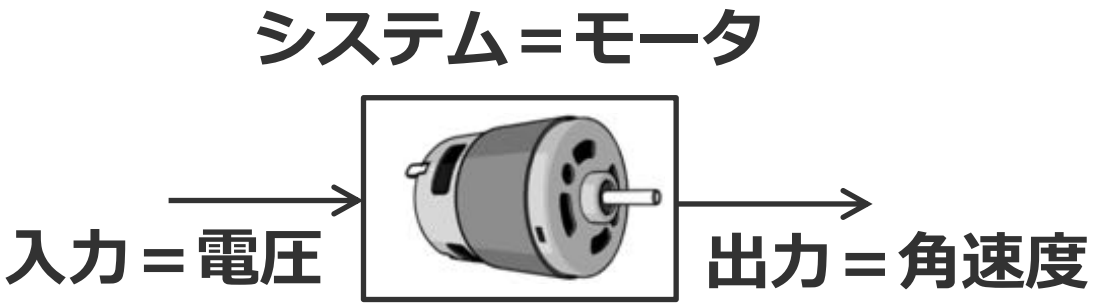
$$\boldsymbol{A}=\begin{bmatrix}0 & 1 & 0 & \cdots & 0\\ \vdots & \ddots & \ddots & \ddots & \vdots\\ \vdots & & \ddots & \ddots & 0\\ 0 & \cdots & \cdots & 0 & 1\\ -a_0 & -a_1 & \cdots & \cdots & -a_{n-1}\end{bmatrix}\quad \boldsymbol{B}=\begin{bmatrix}0\\0\\\vdots\\0\\0\\1\end{bmatrix}$$

$$\boldsymbol{C}=\begin{bmatrix}b_0 & \cdots & b_m & 0 & \cdots & 0\end{bmatrix}$$

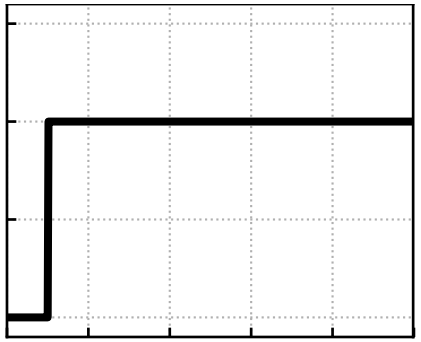
$$P(s)=\frac{b_m s^m+b_{m-1}s^{m-1}+\cdots+b_1 s+b_0}{s^n+a_{n-1}s^{n-1}+\cdots+a_1 s+a_0}$$

可制御正準形での実現（最小実現）

モデルの特徴を知る



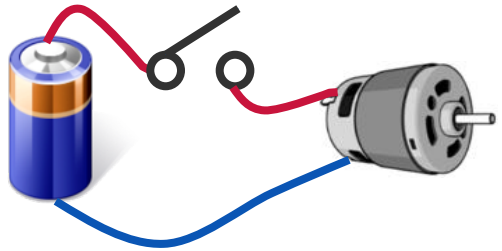
入力を加えて出力を観測する
→ 特徴を調べる

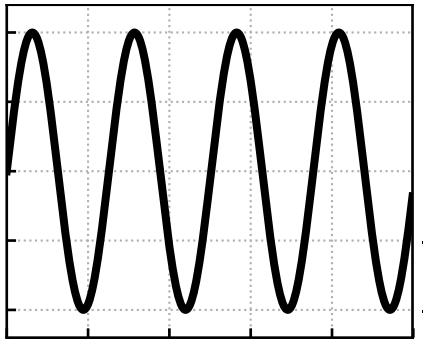


ステップ入力

電池をつないだときの出力の振る舞いは？

→ システムのステップ応答（過渡・定常特性）

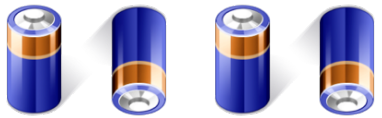




正弦波入力

電池の＋-を交互に切り替えるとどんな振る舞いになるか？

→ システムの周波数応答（ゲイン，位相）



Python演習２

伝達関数モデルのステップ応答

以下のコードを実行してみましょう

$$P(s) = \frac{1}{0.5s+1} = \frac{0s+1}{0.5s+1}$$

```
from control.matlab import *
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots(figsize=(3, 2.3))

P = tf([0, 1], [0.5, 1])

y, t = step(P, np.arange(0, 5, 0.01))
ax.plot(t, y, color='k')

ax.set_xticks(np.linspace(0, 5, 6))
ax.grid(ls=':’)
stepinfo(P)
```

Python ：伝達関数モデルの定義

```
tf( [分子の係数], [分母の係数] )
```

Python ：ステップ応答

```
y, t = step(モデル, 時間)
```

出力 時間

Python ：ステップ応答の情報

```
Info = stepinfo(モデル)
```

Python演習２

伝達関数モデルのステップ応答

以下のコードを実行してみましょう

$$P(s) = \frac{1}{0.5s+1} = \frac{0s+1}{0.5s+1}$$

```
P = tf([0, 1], [0.5, 1])
P
```

$$\frac{1}{0.5s+1}$$

←TeXの表示になります
　右クリックするとTeXコマンドが取得可

```
s = tf('s')
P = 1/(0.5*s + 1)
P
```

$$\frac{1}{0.5s+1}$$

←こういう書き方もできます

Python演習２

伝達関数モデルの周波数応答

以下のコードを実行してみましょう

$$P(s) = \frac{1}{0.5s+1} = \frac{0s+1}{0.5s+1}$$

Python : 周波数応答

gain, phase, w = bode(モデル, 周波数域)

ゲイン(倍) 位相（rad） 周波数（rad/s）

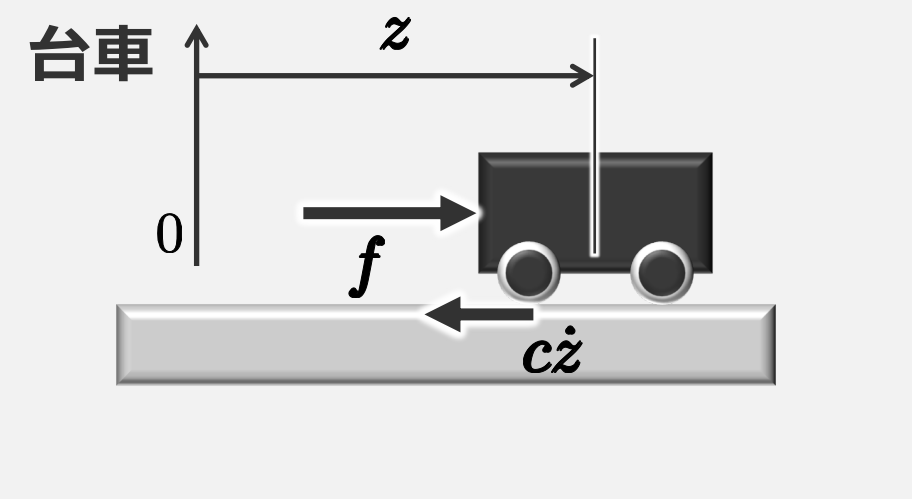
```
K = 1
T = 0.5

fig, ax = plt.subplots(2,1,figsize=(4,3.5))

P = tf([0, K],[T, 1])
gain, phase, w = bode(P, logspace(-2,2), Plot=False)
ax[0].semilogx(w, 20*np.log10(gain))
ax[1].semilogx(w, phase*180/np.pi)

ax[0].grid(which="both", ls=':')
ax[0].set_ylabel('Gain [dB]')
ax[1].grid(which="both", ls=':')
ax[1].set_xlabel('$¥omega$ [rad/s]')
ax[1].set_ylabel('Phase [deg]')
```

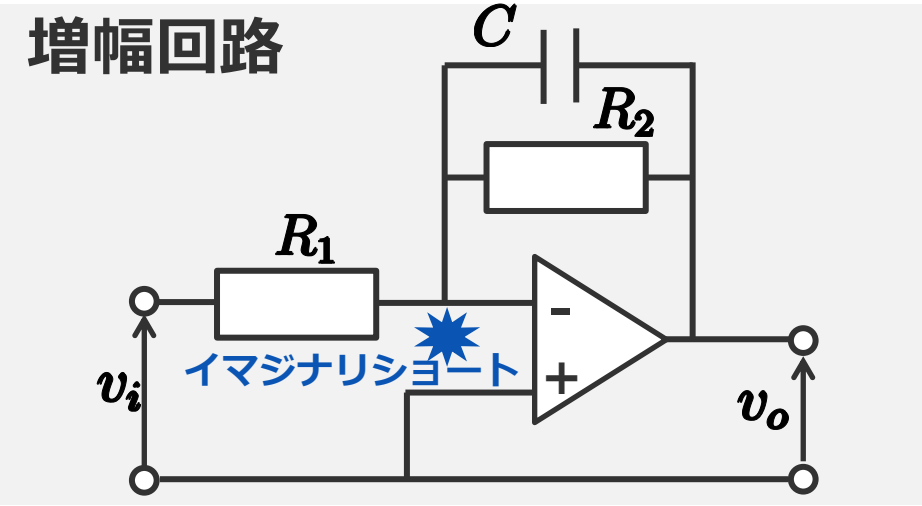
0.01 ~ 100 → logspace(-2,2)

dBへの変換：20*np.log10(gain)

degへの変換：phase*180/np.pi

1次遅れ系





$$\frac{1}{ms+c}$$

$$-\frac{R_2}{R_1R_2Cs+R_1}$$

1次遅れ系：$G(s)=\dfrac{K}{1+Ts}$

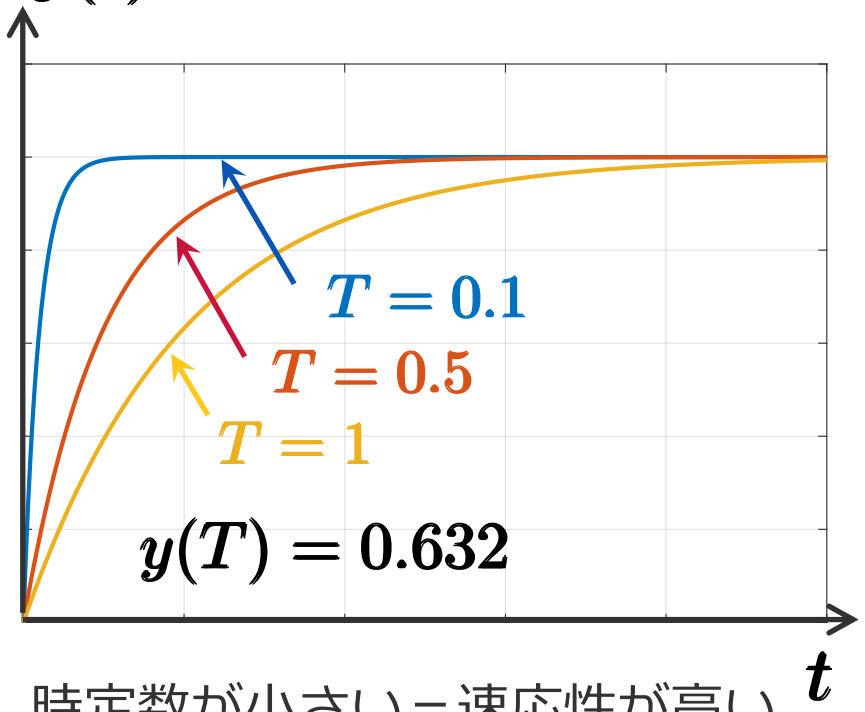
T ：時定数

K ：ゲイン

1次遅れ系

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

$$y(t) = 1 - e^{-\frac{t}{T}}$$

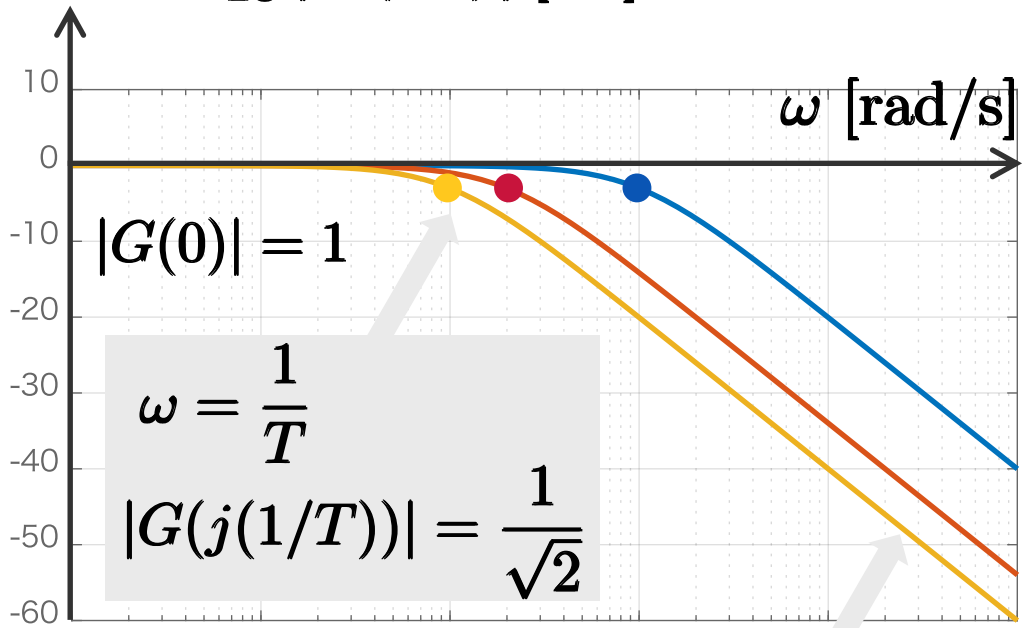


時定数が小さい＝速応性が高い

振動しない→オーバーシュートなし

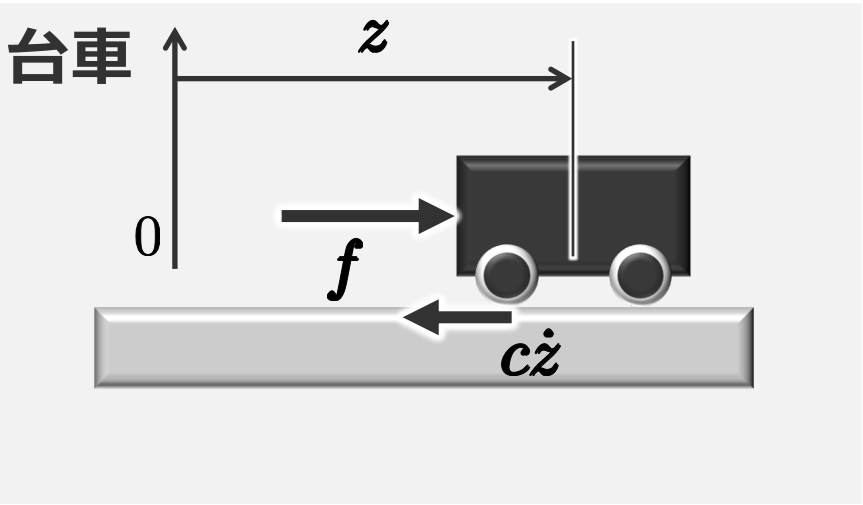
$$G(j\omega) = \frac{1}{1+j\omega T}$$

$20\log_{10}|G(j\omega)|$ [dB]



$$|G(j\omega)| \simeq \frac{1}{\omega T} \rightarrow \text{-20[dB/dec]}$$

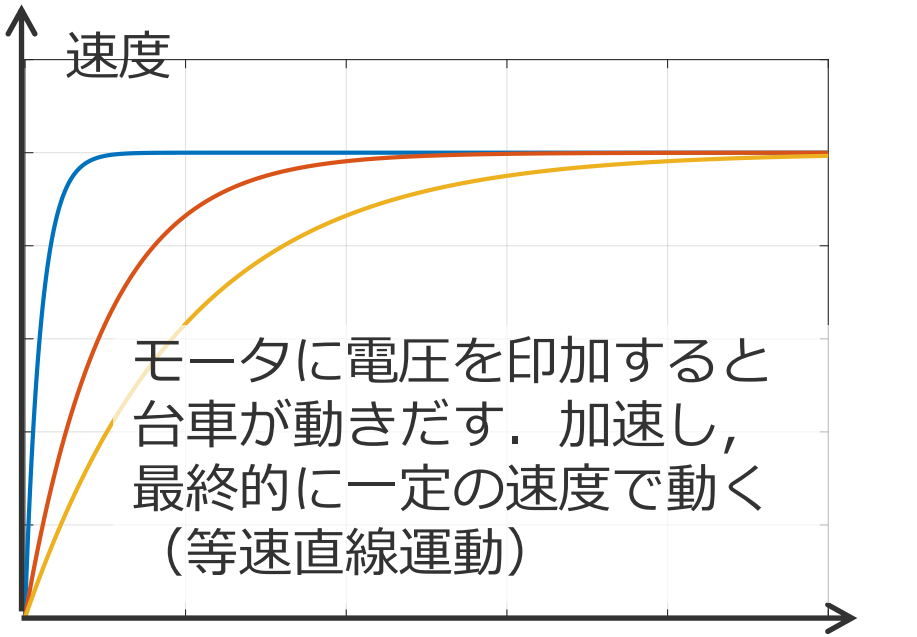
1次遅れ系

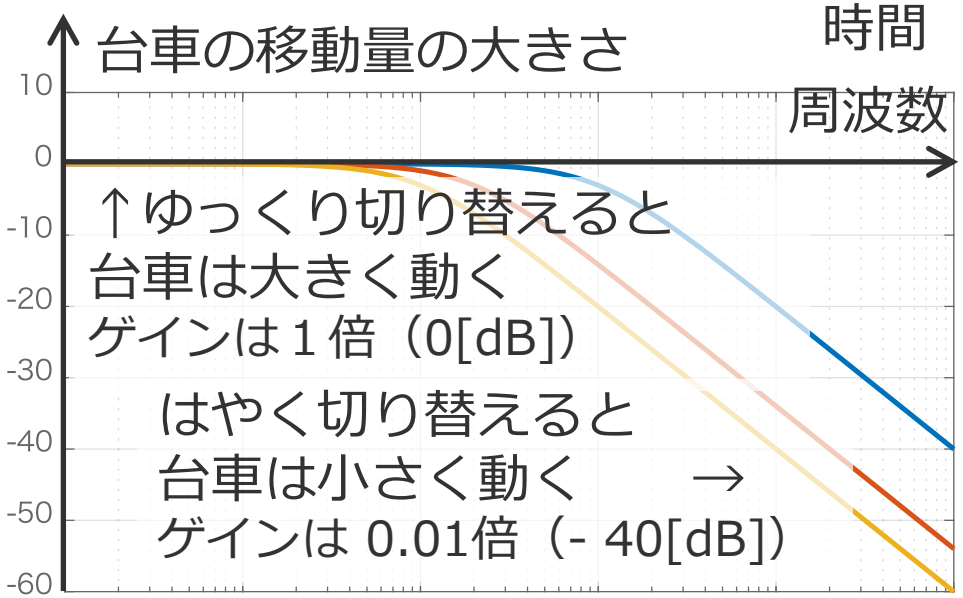


$$\frac{1}{ms+c} = \frac{K}{1+Ts}$$

$$T = \frac{m}{c} \quad K = \frac{1}{c}$$

モータの正回転・逆回転を
交互に切り替えるとどうなるか？



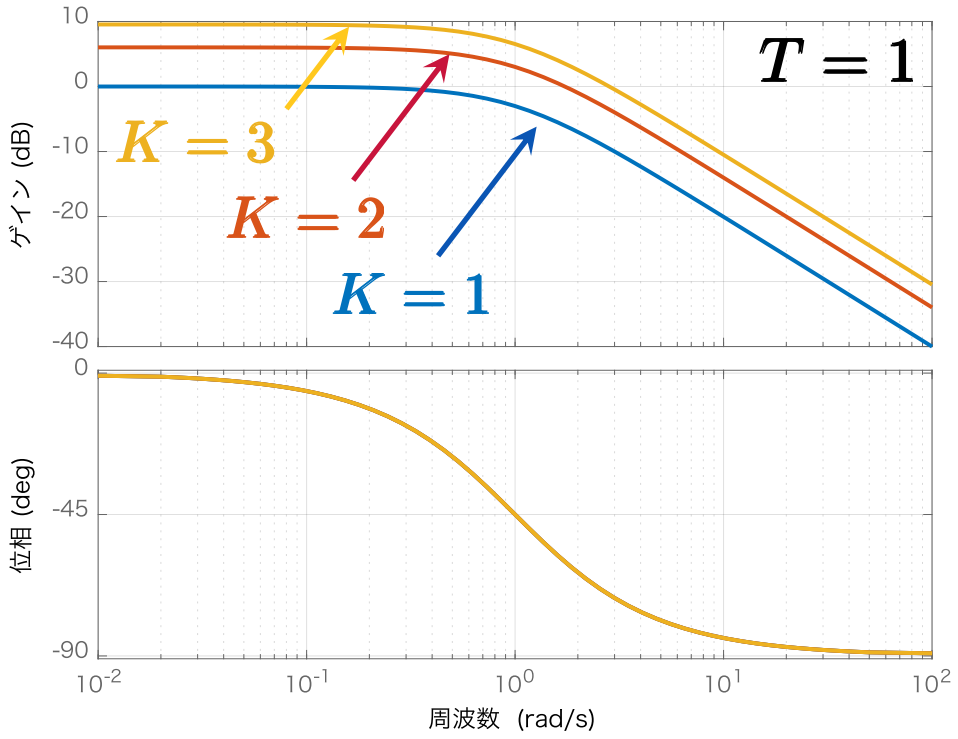


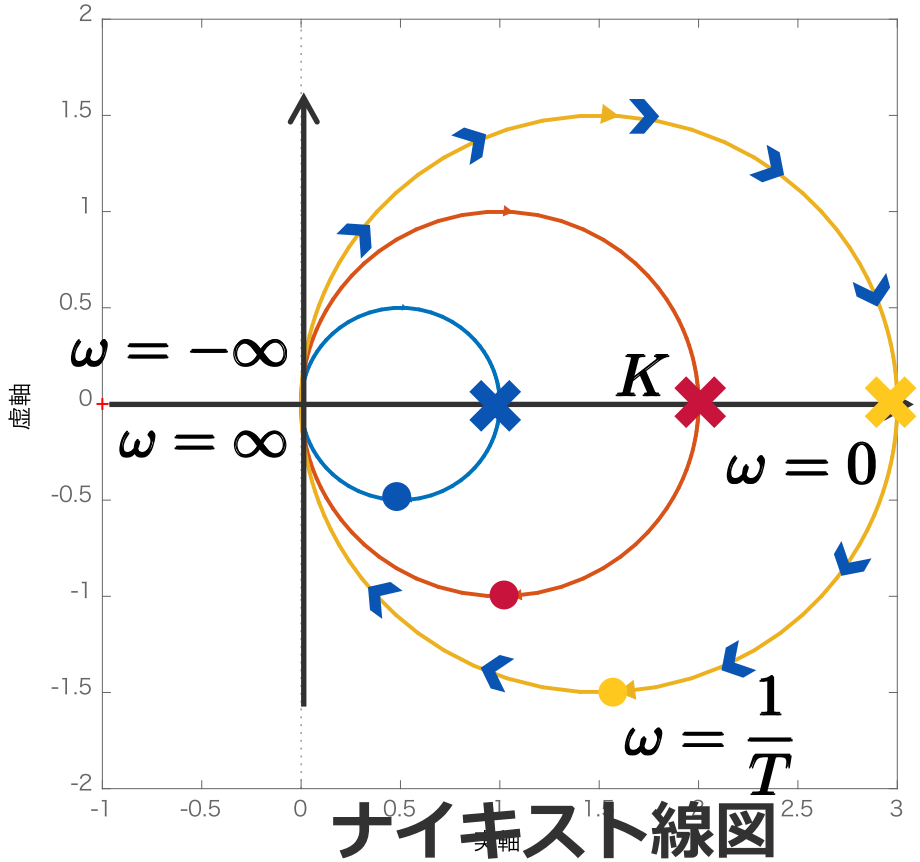
1次遅れ系

$$G(j\omega) = \frac{K}{1 + j\omega T}$$

ゲイン $|G(j\omega)| = \dfrac{K}{\sqrt{1 + (\omega T)^2}}$

位相 $\angle G(j\omega) = -\tan^{-1}(\omega T)$

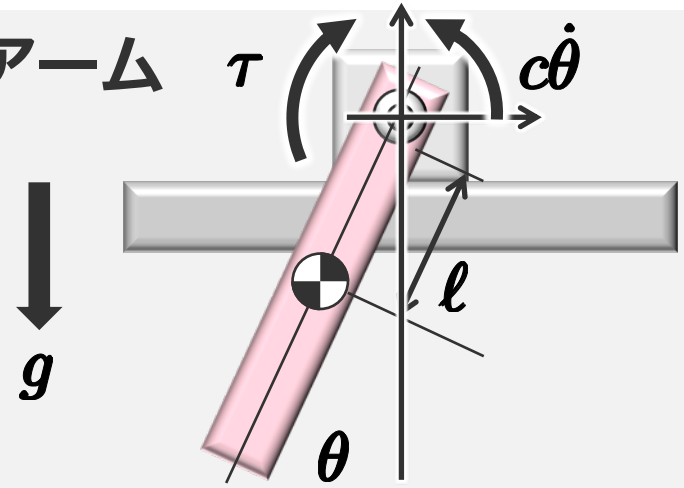




ナイキスト線図

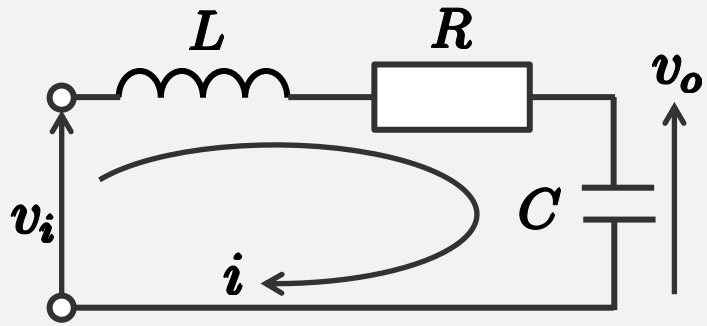
２次遅れ系

垂直アーム



$$\frac{1}{Js^2+cs+mg\ell}$$

RCL回路



$$\frac{1}{CLs^2+CRs+1}$$

２次遅れ系：$G(s)=\dfrac{K\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}$

ζ ：減衰係数

ω_n：固有角周波数

K ：ゲイン

2次遅れ系

マスバネダンパ

$\omega_n = \sqrt{k/m}$

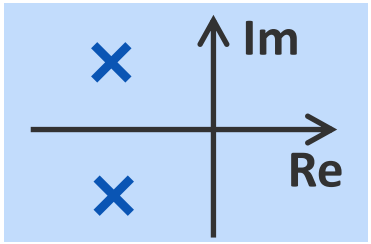
$\zeta = c/(2\sqrt{mk})$

$K = 1/k$

$$G(s) = \frac{K\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} = \frac{p_1 p_2}{(s-p_1)(s-p_2)}$$

減衰係数　固有角周波数

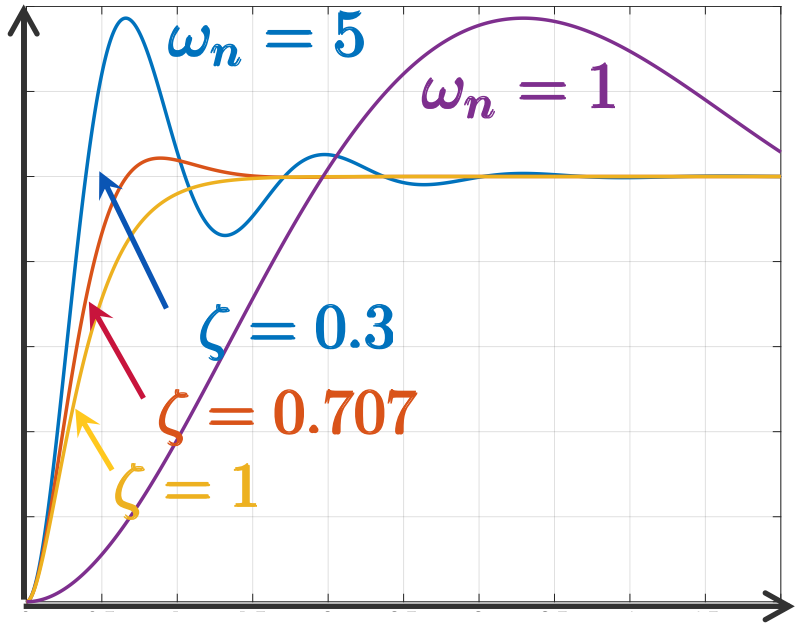
$p_1, p_2 = \alpha \pm \beta j$ のとき



$$G(s) = \frac{\alpha^2 + \beta^2}{s^2 - 2\alpha s + \alpha^2 + \beta^2}$$

$$\begin{cases} K = 1 \\ \zeta = -\dfrac{\alpha}{\sqrt{\alpha^2+\beta^2}} \quad 0 \le \zeta \le 1 \\ \omega_n = \sqrt{\alpha^2+\beta^2} \end{cases}$$

ステップ応答



虚部が大きい ＝ ζ が小さい ＝ 振動的になる

実部or虚部が大きい ＝ ω_n が大きい ＝ 応答が速い

2次遅れ系

マスバネダンパ

$\omega_n = \sqrt{k/m}$

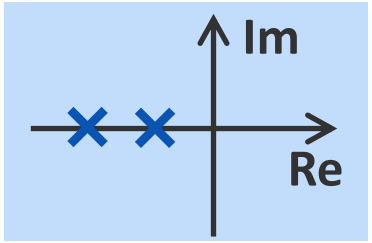
$\zeta = c/(2\sqrt{mk})$

$K = 1/k$

$$G(s) = \frac{K\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} = \frac{p_1 p_2}{(s-p_1)(s-p_2)}$$

減衰係数　固有角周波数

$p_1, p_2 = \alpha_1, \alpha_2$ のとき



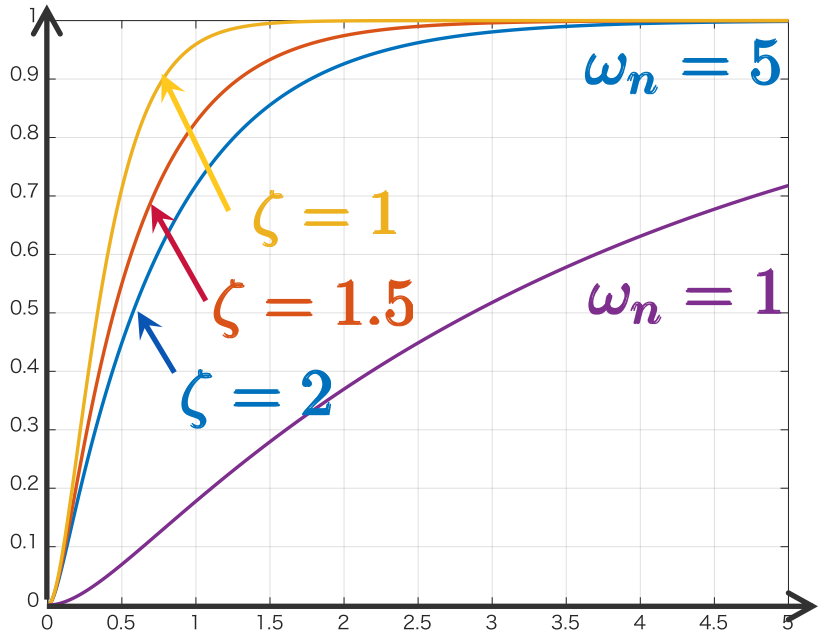
$$G(s) = \frac{\alpha_1\alpha_2}{s^2 - (\alpha_1+\alpha_2)s + \alpha_1\alpha_2}$$

$$\begin{cases} K = 1 \\ \zeta = -\dfrac{\alpha_1+\alpha_2}{2\sqrt{\alpha_1\alpha_2}} \\ \omega_n = \sqrt{\alpha_1\alpha_2} \end{cases}$$

←相加>相乗

$\zeta \geq 1$ で，振動しない．極が負側に大きくなると，ζ と ω_n が大きくなる = 制動が強くなる & 応答が速くなる

ステップ応答

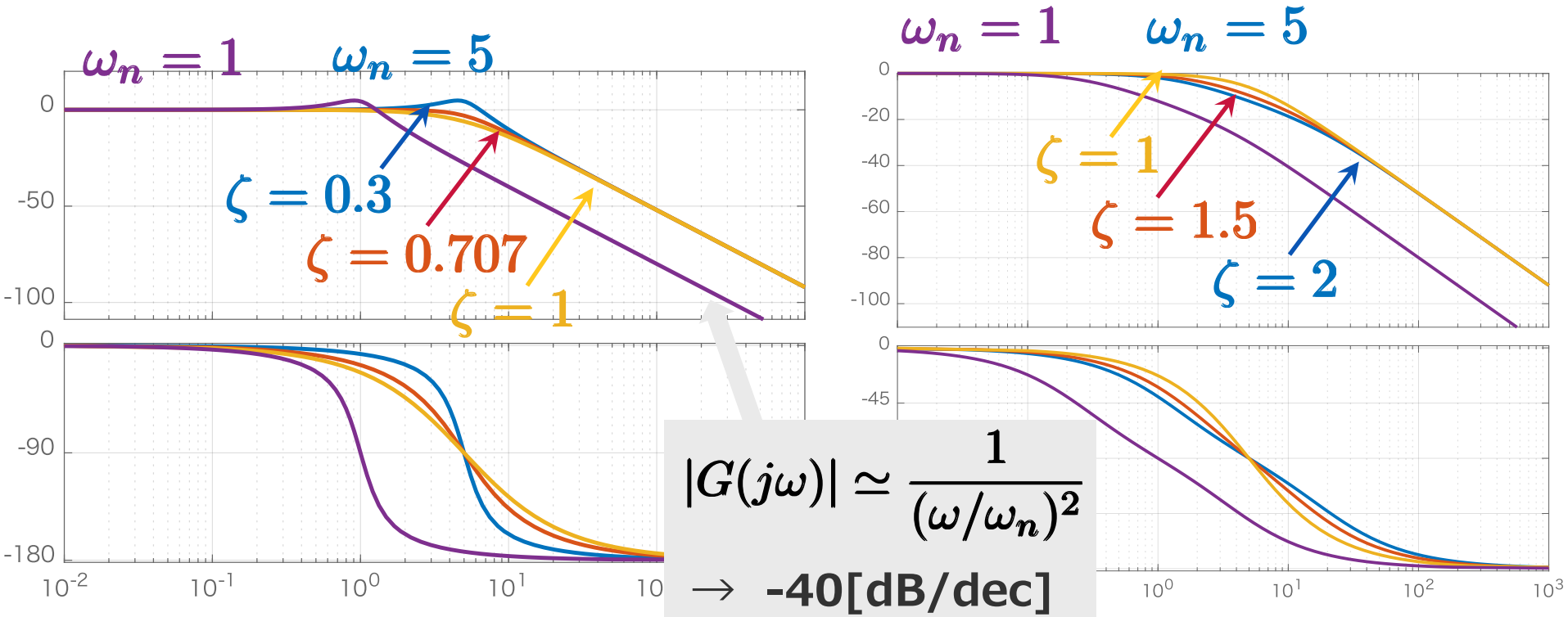


2次遅れ系

周波数応答

$$G(j\omega) = \frac{\omega_n^2}{\omega_n^2 - \omega^2 + j2\zeta\omega_n\omega}$$

$$|G(j\omega)| = \frac{\omega_n^2}{\sqrt{(\omega_n^2 - \omega^2)^2 + (2\zeta\omega_n\omega)^2}} \quad \angle G(j\omega) = -\tan^{-1}\frac{2\zeta\omega_n\omega}{\omega_n^2 - \omega^2}$$

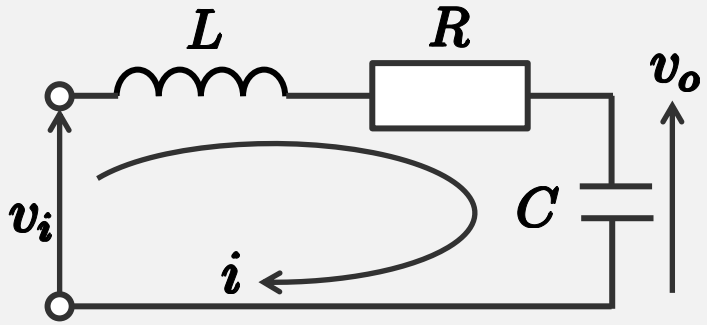


$$|G(j\omega)| \simeq \frac{1}{(\omega/\omega_n)^2}$$

$\rightarrow$ **-40[dB/dec]**

2次遅れ系

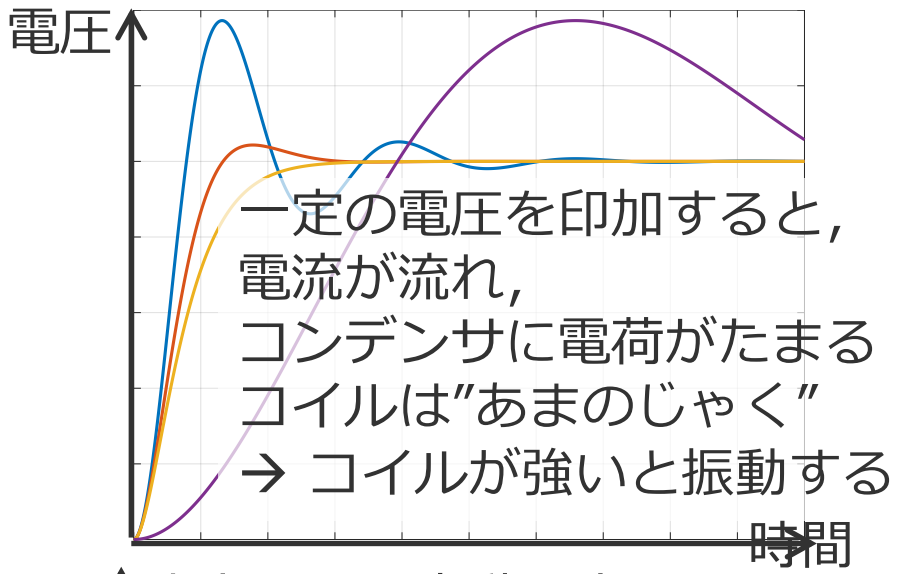
RCL回路

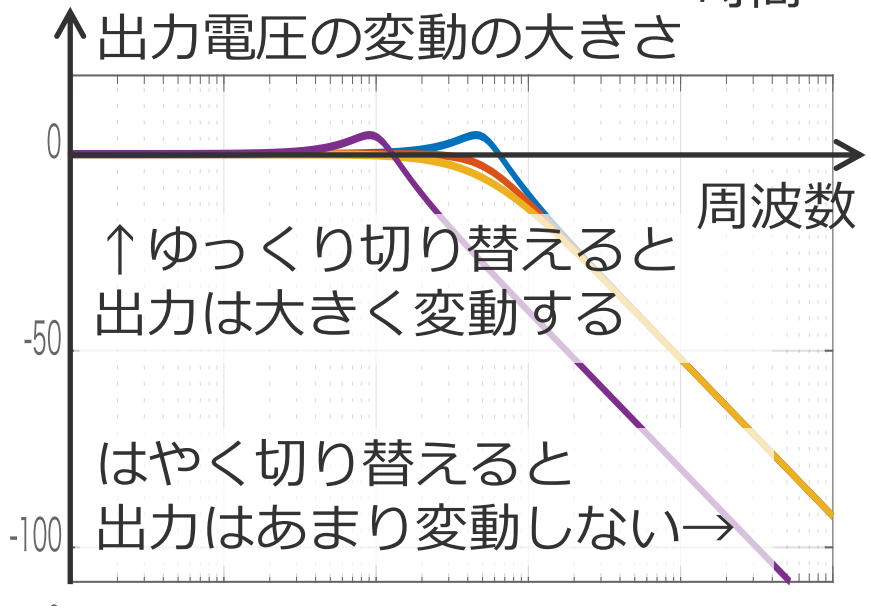


$$\frac{1}{CLs^2+CRs+1}$$

$$\omega_n=\sqrt{\frac{1}{CL}} \qquad \zeta=\frac{R\sqrt{C}}{2\sqrt{L}}$$

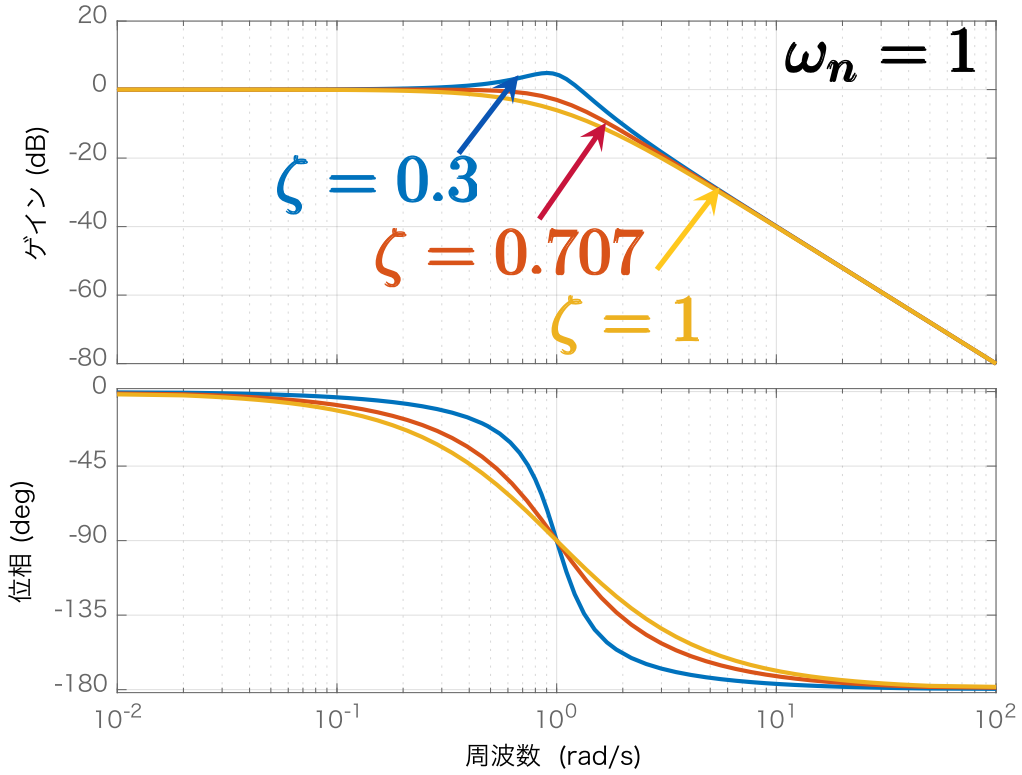
充電と放電を切り替える

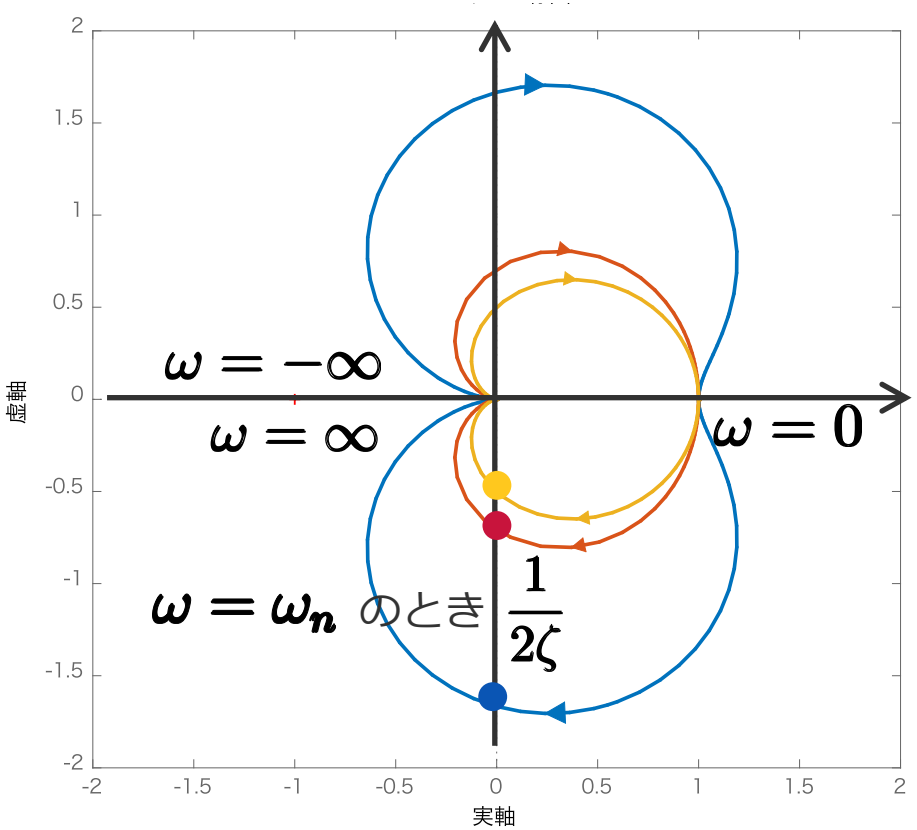




2次遅れ系

$$|G(j\omega)| = \frac{\omega_n^2}{\sqrt{(\omega_n^2 - \omega^2)^2 + (2\zeta\omega_n\omega)^2}} \qquad \angle G(j\omega) = -\tan^{-1}\frac{2\zeta\omega_n\omega}{\omega_n^2 - \omega^2}$$



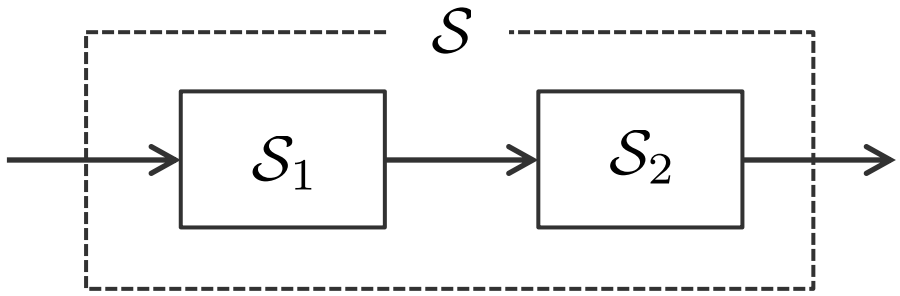


ナイキスト線図

ピークゲイン $M_r = \dfrac{1}{2\zeta\sqrt{1-\zeta^2}} \quad 0 \le \zeta \le \dfrac{1}{\sqrt{2}}$

ブロック線図

直列結合

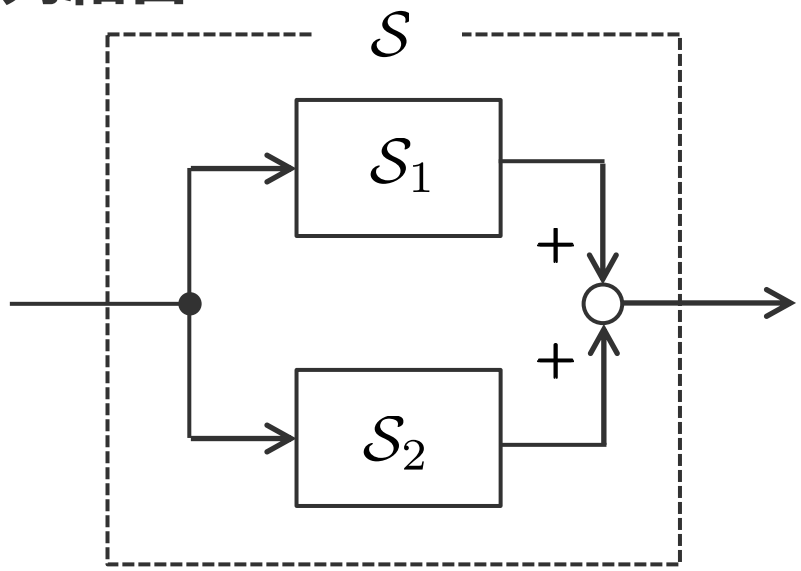


$$\mathcal{S} = \mathcal{S}_1 \mathcal{S}_2$$

Python ：直列結合

```
series(モデル1, モデル2)
モデル1 * モデル2
```

並列結合

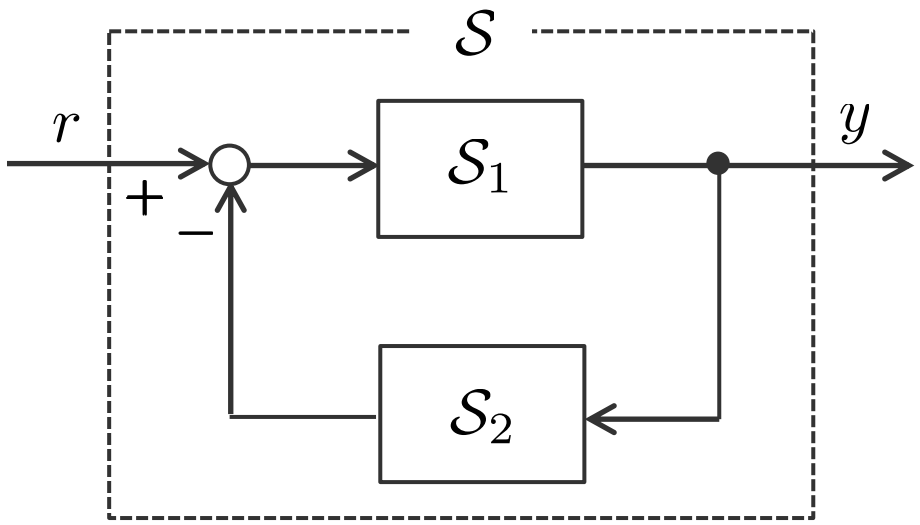


$$\mathcal{S} = \mathcal{S}_1 + \mathcal{S}_2$$

Python ：直列結合

```
parallel(モデル1, モデル2)
モデル1 + モデル2
```

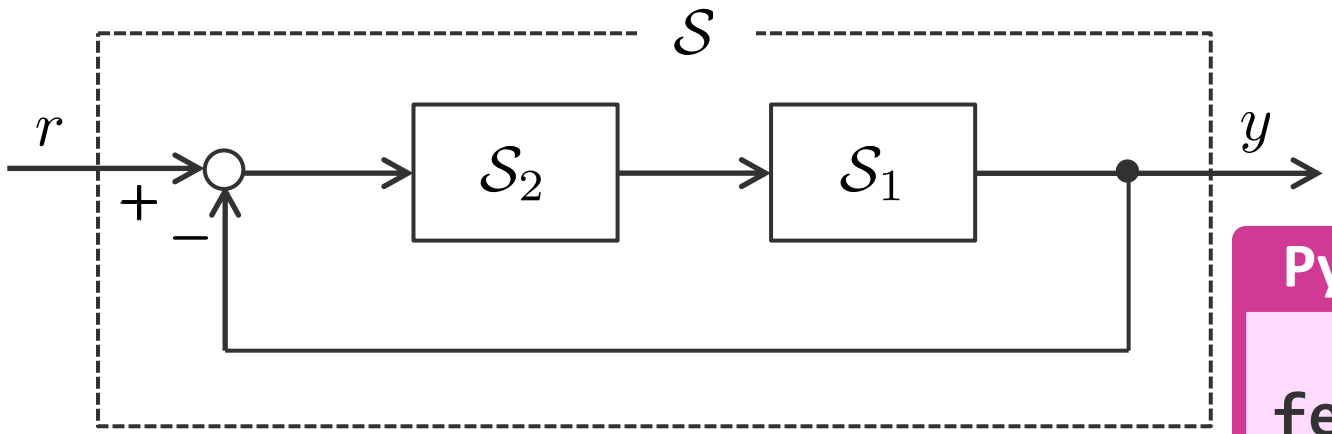
ブロック線図



$$\mathcal{S} = \frac{\mathcal{S}_1}{1 + \mathcal{S}_1\mathcal{S}_2}$$

Python：フィードバック結合

```
feedback(モデル1, モデル2, sign = -1)
```



$$\mathcal{S} = \frac{\mathcal{S}_1\mathcal{S}_2}{1 + \mathcal{S}_1\mathcal{S}_2}$$

Python：フィードバック結合

```
feedback(モデル1*モデル2, 1, sign = -1)
```

HomeWork

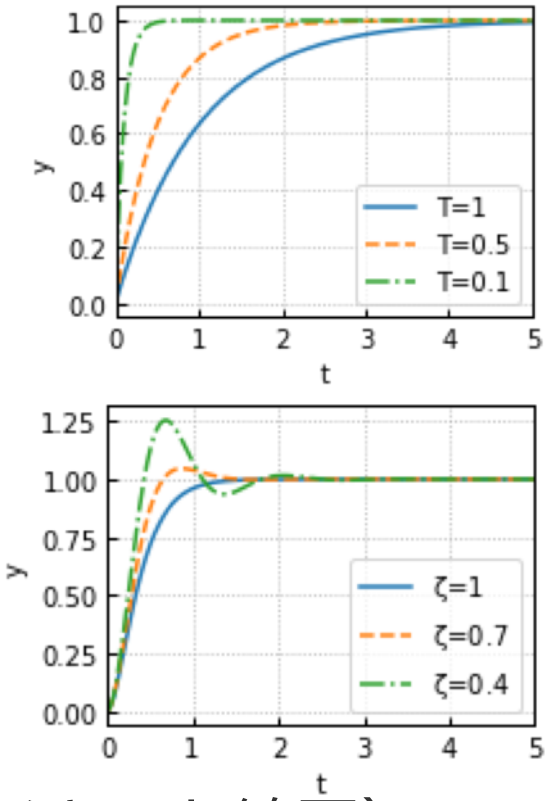
★グラフを描いて考察しよう

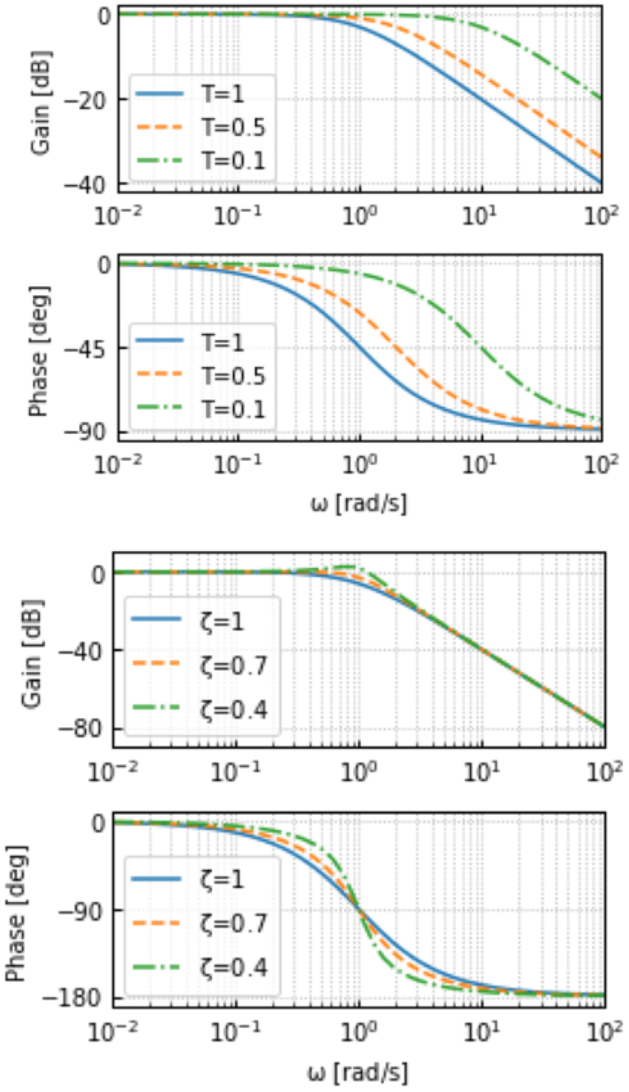
1 次遅れ系のステップ応答

1 次遅れ系の周波数応答

2 次遅れ系のステップ応答

2 次遅れ系の周波数応答





（1 次遅れ系，2 次遅れ系のナイキスト線図）

★手計算に挑戦してみよう

ステップ応答や周波数応答を泥臭く計算

今日の目標

Level☆☆★

- 制御とはなにかを説明できる (14~25)
- 伝達関数モデルのステップ応答のグラフをPythonで描ける (71)
- 伝達関数モデルの周波数特性のグラフをPythonで描ける (73)

Level☆★★

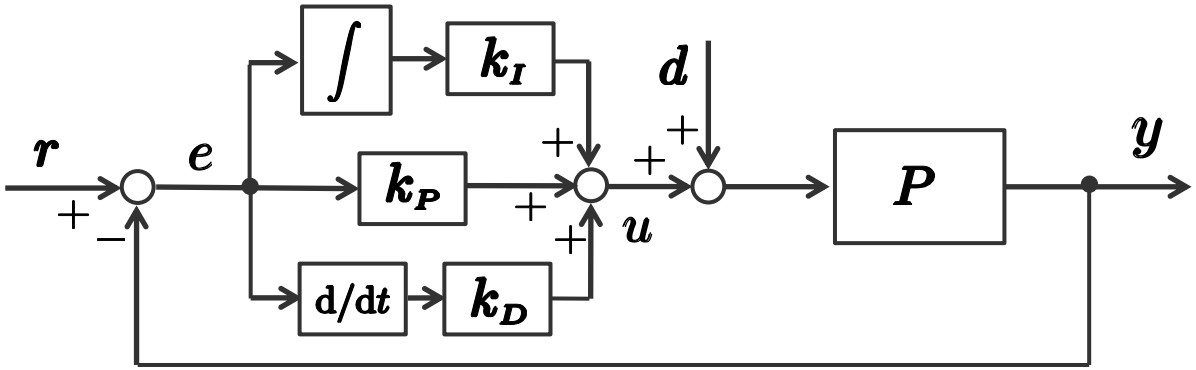
- 伝達関数モデルを作ることができる (65~68)
- １次遅れ系，２次遅れ系の特性を理解できる (74~83)

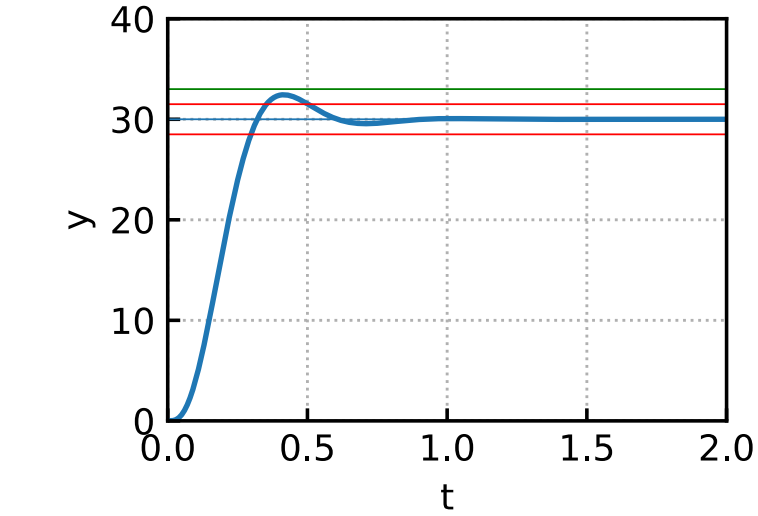
Level★★★

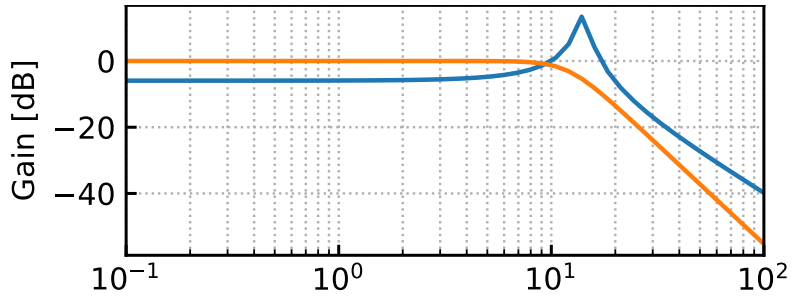
- 設計モデルの種類とそれらの関係を理解できる (63)
- ステップ応答や周波数応答を手計算で求めることができる (参考図書)

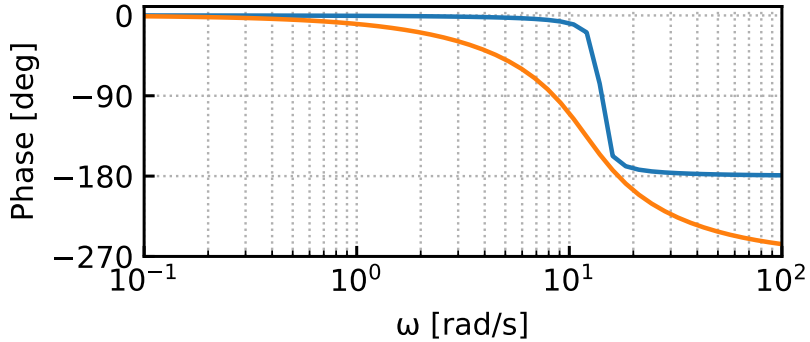
次回予告（6/9）


6月
9
【オンライン勉強会】Pythonで学ぶ制御工学 Part2
YouTubeLive によるオンライン勉強会．初学者歓迎！
主催：SICE関西支部見学会委員会
SICE関西支部主催
オンライン勉強会
Pythonで学ぶ制御工学
✓ Part1 Pythonと制御工学の基礎
✓ Part2 伝達関数モデルを用いた制御系設計
✓ Part3 ループ整形，現代制御理論の基礎


r
e
∫
k_I
k_P
d/dt
k_D
d
u
P
y


y
t


Gain [dB]


Phase [deg]
ω [rad/s]