

# インタラクティブシステム論 第8回

梶本裕之



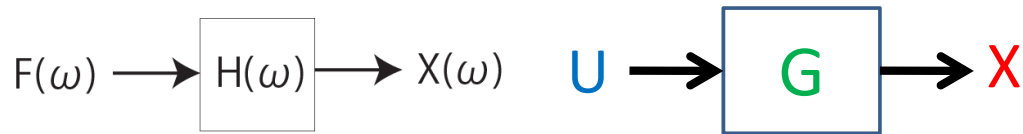
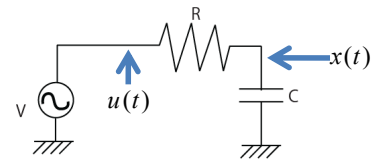
## 日程

講義番号	講義日	講義内容	pdf	video	レポート締め切り
1	4/8	イントロダクション	[  pdf ] 2021年版	<a href="#">video</a>	4/15
		Scilab課題	[  pdf ](更新なし)		↑
		上記資料のPython版	[  pdf ](更新なし)		↑
2	4/15	フーリエ変換	[  pdf ] 2021年版	<a href="#">video</a>	4/22
3	4/22	フーリエ変換と線形システム	[  pdf ] 2021年版	<a href="#">video</a>	4/29
-	4/29	昭和の日			
4	5/6	信号処理の基礎	[  pdf ] 2021年版	<a href="#">video</a>	5/13
5	5/13	信号処理の応用1(相関)	[  pdf ] 2021年版	<a href="#">video</a>	5/20
6	5/20	信号処理の応用2(画像処理)	[  pdf ] 2021年版	<a href="#">video</a>	5/27
-	5/27	(出張) 中間確認テスト準備 (自習)			
-	6/3	(出張) 中間確認テスト準備 (自習)			
-	6/10	中間確認テスト			
7	6/17	ラプラス変換	[  pdf ] 2021年版	<a href="#">video</a>	6/24
8	6/24	古典制御の基礎	[  pdf ] 2021年版	<a href="#">video</a>	7/1
9	7/1	行列	[  pdf ] 2021年版	<a href="#">video</a>	7/8
10	7/8	行列と最小二乗法	[  pdf ] 2021年版	<a href="#">video</a>	7/15
11	7/15	ロボティクス	[  pdf ] 2021年版	<a href="#">video</a>	7/22
-	7/22	期末テスト準備 (自習)	[  pdf ]2021年版		
-	7/29	期末確認テスト (現在は大学を予定)			

日程およびテストを大学で行うかについては、随時アナウンスします。Google Classroomでもアナウンスの予定。



## 前回の話まとめ



フーリエ変換で扱えない、非定常な状況をラプラス変換で扱う  
システムの応答は次のようなステップで求めることができる

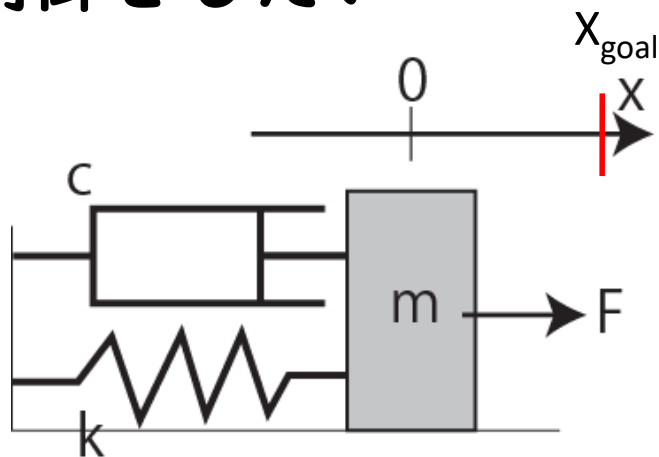
- (1) システムの入出力関係を決める **伝達関数G** をもとめる
- (2) **入力のラプラス変換U** を求める
- (3)  $X=GU$  によって **出力のラプラス変換X** が得られる。
- (4) 逆ラプラス変換によって **出力波形** が得られる。



# 制御の基礎の基礎

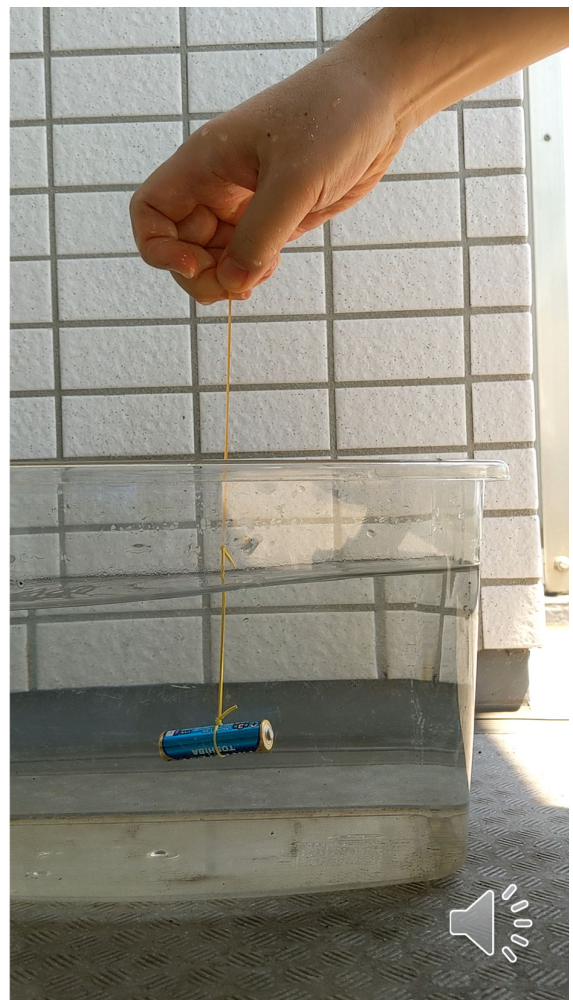


# 制御をしたい



時刻0に、 $x=0$ の位置にあったおもり $m$ を、 $x=x_{\text{goal}}$ に移動したい。

どのような力 $F(t)$ を加えたらよいだろうか？

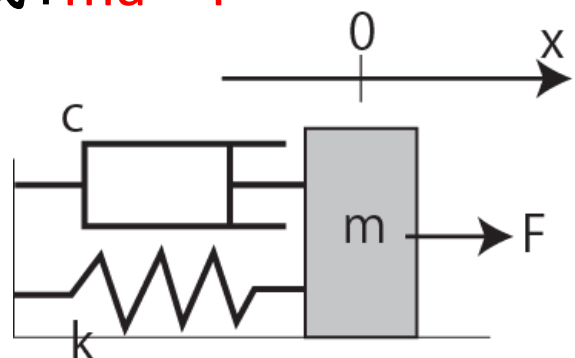


# モデルを作る

ニュートンの運動方程式:  $ma = F$

おもりに加わる力

- 外力(制御入力):  $f$
- ダンパ:  $-cv$
- バネ:  $-kx$



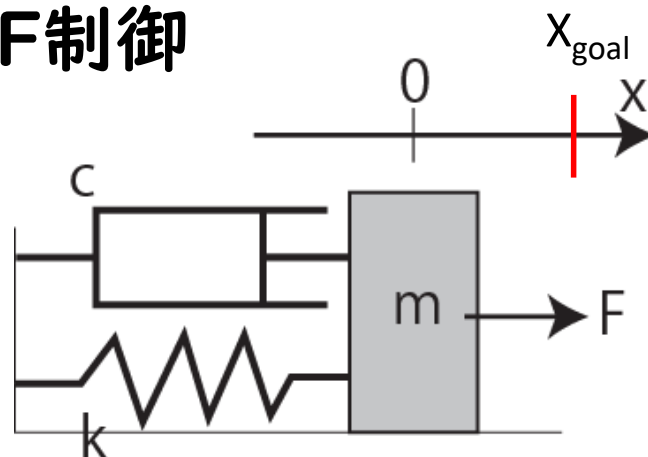
運動方程式:

2階微分まで考えるシステム=2次系

多くのシステムの近似モデルとして適用可能。



# ON・OFF制御



- 一番初めに考える制御
- 目的の位置より手前だったら  $F=1$  を加える.
- 目的の位置を超えたら  $F=0$  とする.



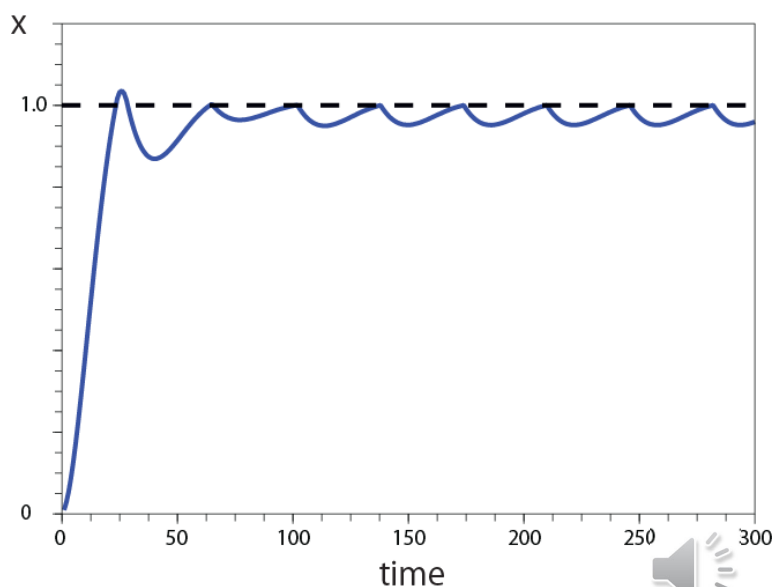
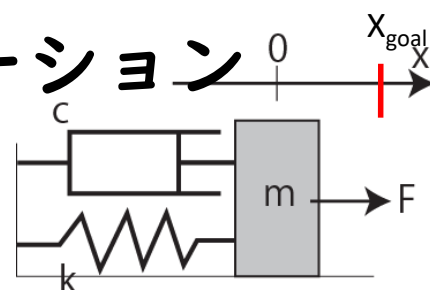
## ON/OFF制御のシミュレーション

Scilabコード

```
m=1.0;           //質量
c=1.0;           //ダンパ
k=1.0;           //バネ
xgoal=1.0;       //目標値
x=0;             //現在地
v=0;            //現在速度
dt = 0.1;        //時間間隔
xrecord = [];    //データ記録用
for t=1:300,
```

```
    a = (F - k*x - c*v)/m;
    v = v+a*dt;
    x = x+v*dt;
    xrecord = [xrecord,x];
```

```
end
plot(xrecord);
```



# Scilabでアニメーション

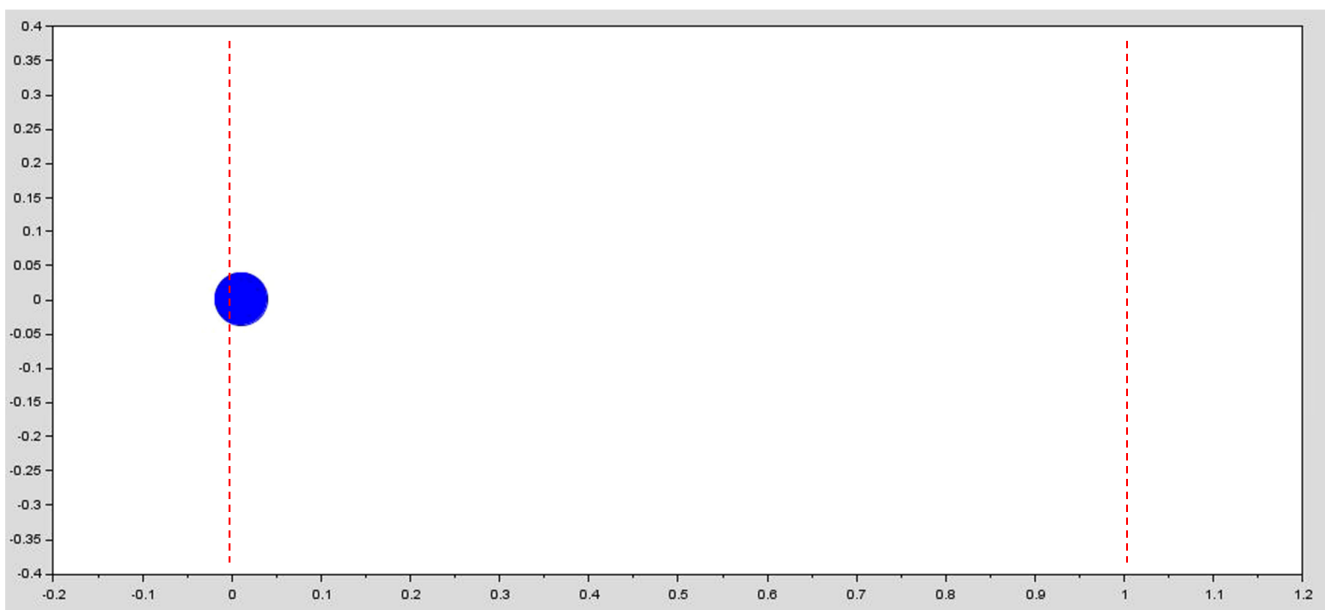
コードを最後に追加

```
// Draw initial figure
figure(1);
plot(xrecord(1),0,'o');
h_compound = gce();
h_compound.children.mark_size = 20;
h_compound.children.mark_background = 2;
h_axes = gca();
h_axes.data_bounds = [-1.5,-1.5;1.5,1.5];

// Animation Loop
i = 1;
while i<=length(xrecord)
    drawlater();
    h_compound.children.data = [xrecord(i),0]; drawnow();
    i = i+1;
    sleep(10);
end
```



## ON・OFF制御



# (補足) pythonコード

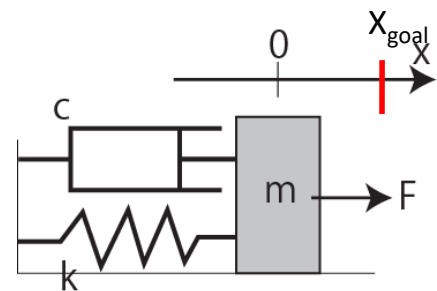
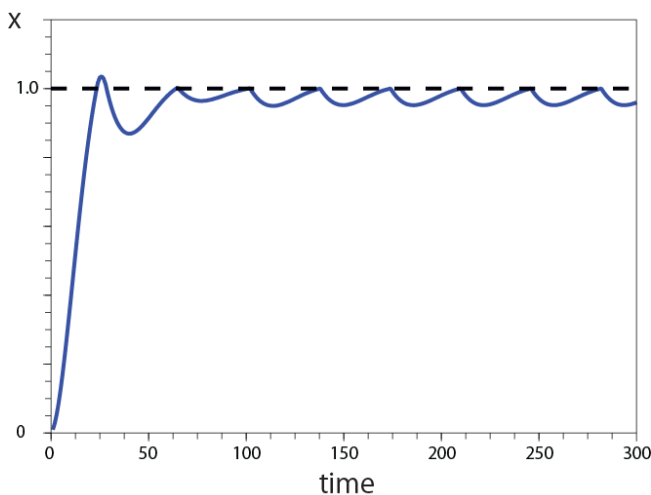
```
import matplotlib.pyplot as plt
import math
import numpy as np
m = 1.0
c = 1.0
k = 1.0
xgoal = 1.0
dt = 0.1
x = 0
v = 0
time = np.arange(0,30.0,dt)
record = list()
for t in time:
    if(x<xgoal):
        F=1.0
    else:
        F=0.0
    a = (F-k*x-c*v)/m
    v = v+a*dt
    x = x+v*dt
    record.append(x)
plt.plot(time,record)
plt.show()
```

アニメーションさせる場合

```
import matplotlib.pyplot as plt
import math
import numpy as np
import matplotlib.animation as animation
fig = plt.figure()
ims = []
m = 1.0
c = 1.0
k = 1.0
xgoal = 1.0
dt = 0.1
x = 0
v = 0
time = np.arange(0,30.0,dt)
record = list()
for t in time:
    if(x<xgoal):
        F=1.0
    else:
        F=0.0
    a = (F-k*x-c*v)/m
    v = v+a*dt
    x = x+v*dt
    im = plt.plot(x,0,marker='o')
    ims.append(im)
ani = animation.ArtistAnimation(fig,ims,interval=10)
plt.show()
```



## ON・OFF制御



•制御の**最低限**は含まれている:

- (1) 対象の状態を見て,
- (2) 目的との差を見て
- (3) 出力を変化させる

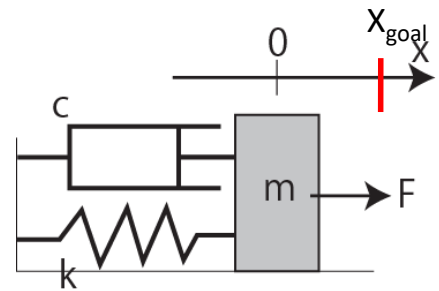
•目標値付近で**永久に発振**してしまう

•ごく簡単なハードウェアで実現できる

•実装例:こたつ(ただしヒステリシスを入れている)



# フィードバックとは



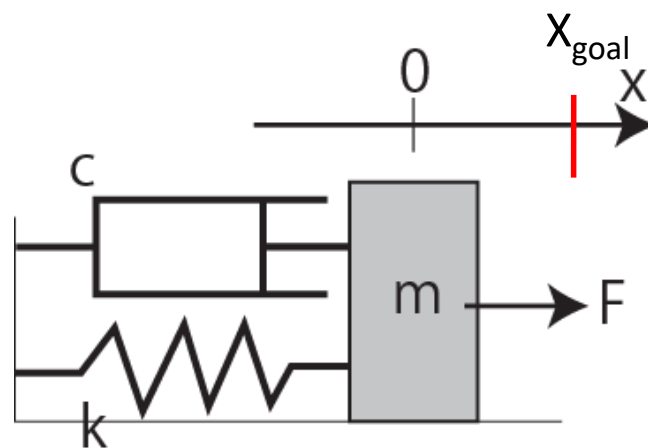
- 制御の**最低限**:

- (1) 対象の状態を見て,
- (2) 目的との差を見て
- (3) 出力を変化させる

- これを, 「フィードバック制御」という



## P制御



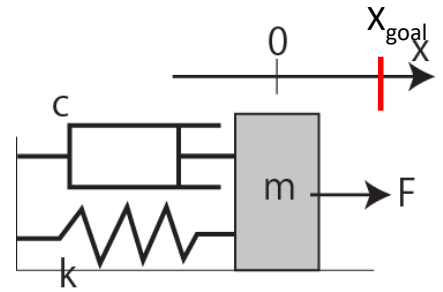
- ON/OFFだと発振してしまった.
- Fをもっと「なだらか」に変化させれば...

- 目標値 $x_{goal}$ と現在値 $x$ との「差」に**比例**した力を加えればよいのでは？

(比例 = **P**roportional)



# P制御のシミュレーション



Scilabコード

```
m=1.0; //質量
c=1.0; //ダンパ
k=1.0; //バネ
xgoal=1.0; //目標値
x=0; //現在値
v=0; //現在速度
dt = 0.1; //時間間隔
xrecord = []; //記録用
```

```
for t=1:300,
```

```
  F=
```

```
  a = (F - k*x - c*v)/m;
```

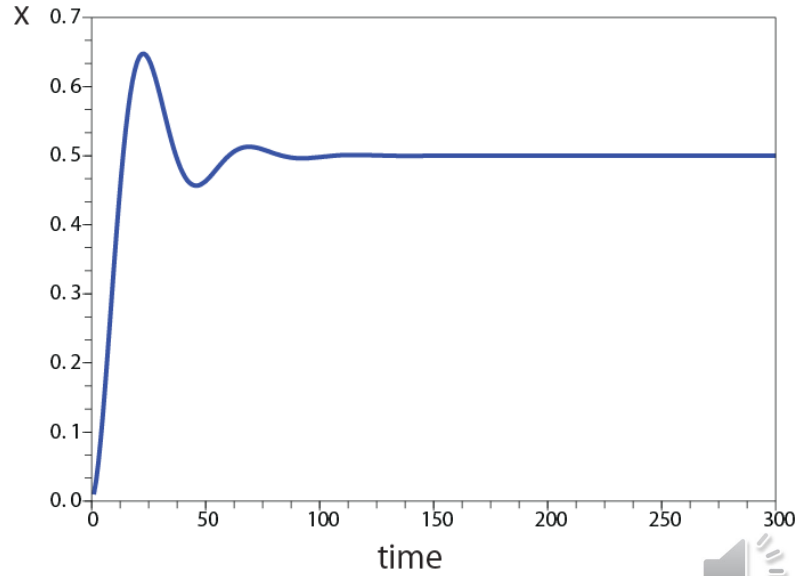
```
  v = v+a*dt;
```

```
  x = x+v*dt;
```

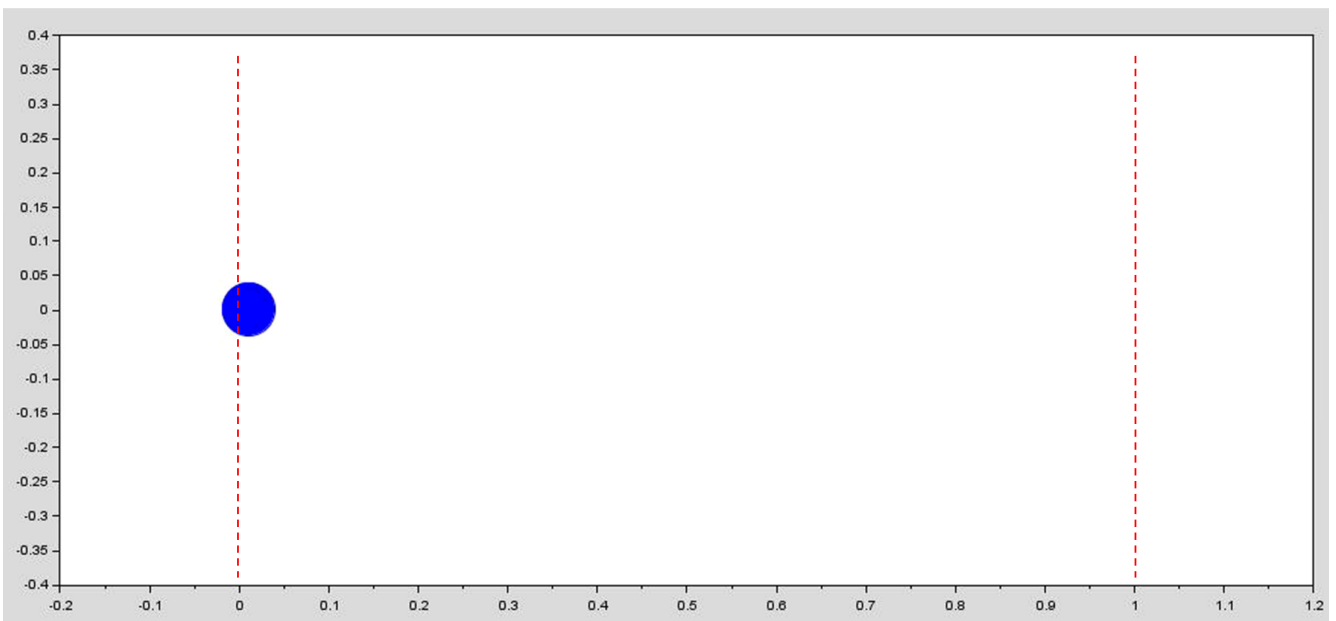
```
  xrecord = [xrecord,x];
```

```
end
```

```
plot(xrecord);
```

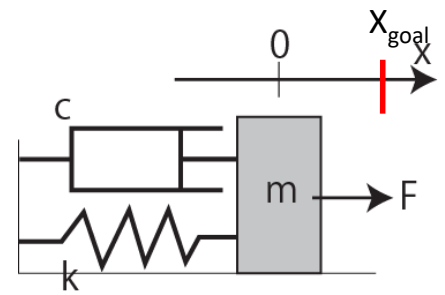
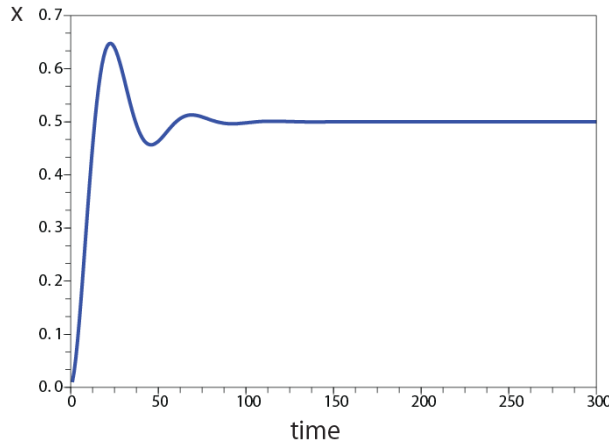


# P制御





# P制御のシミュレーション



- 立ちあがり部分で凹凸
- 目標値 ( $x=1.0$ ) に達していない



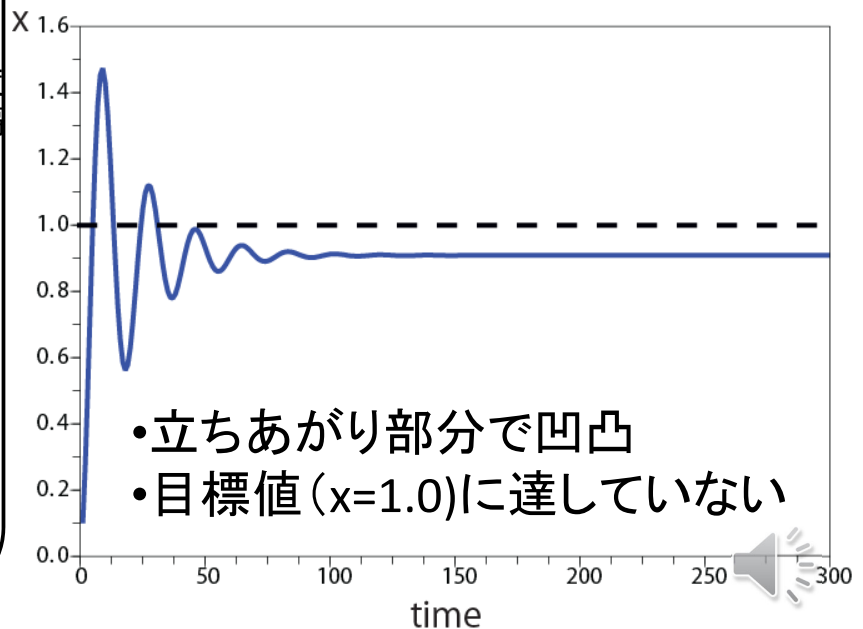
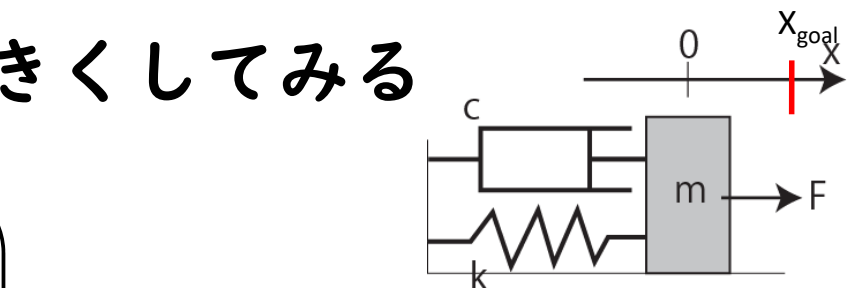
## 比例定数を大きくしてみる

Scilabコード

```
m=1.0; //質量
c=1.0; //ダンパ
k=1.0; //バネ
xgoal=1.0; //目標値
x=0; //現在地
v=0; //現在速度
dt = 0.1; //時間間隔
xrecord = []; //記録用
```

```
for t=1:300,
  F=10.0 * (xgoal-x);
  a = (F - k*x - c*v)/m;
  v = v+a*dt;
  x = x+v*dt;
  xrecord = [xrecord,x];
end
```

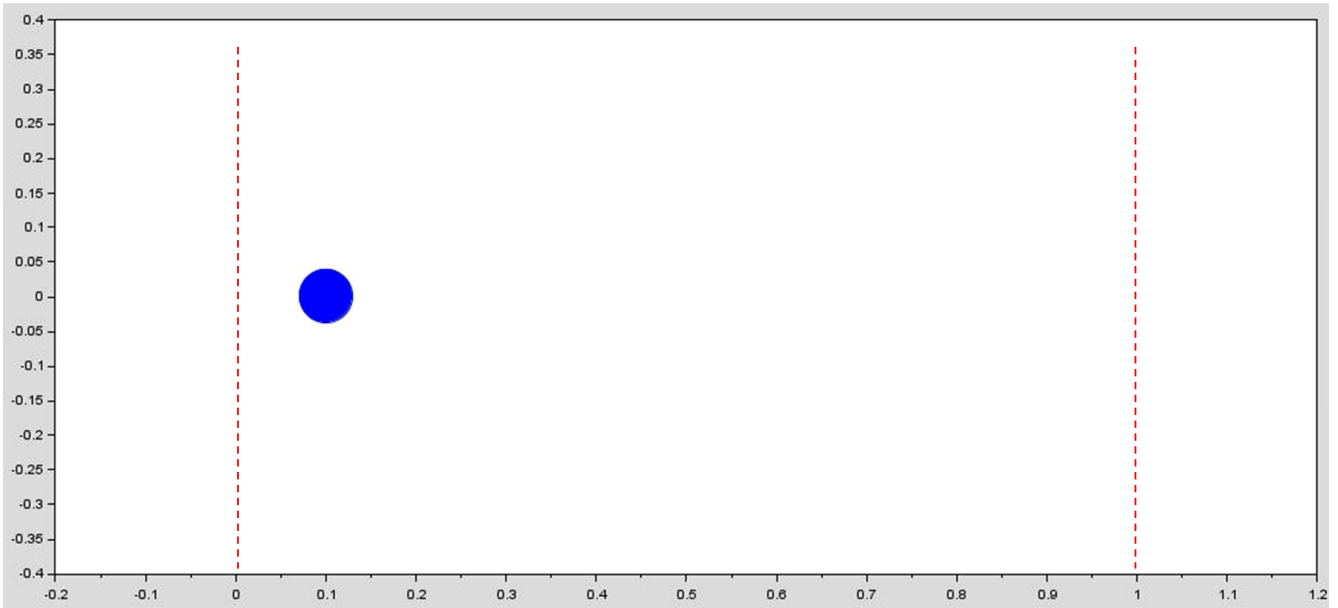
```
plot(xrecord);
```



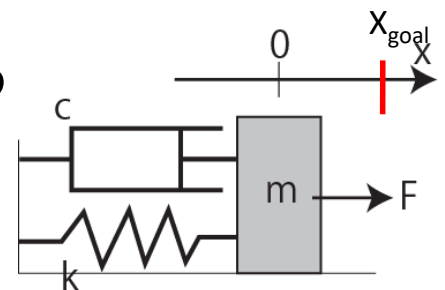
- 立ちあがり部分で凹凸
- 目標値 ( $x=1.0$ ) に達していない



# P制御- ゲイン大



## 比例定数を小さくしてみる

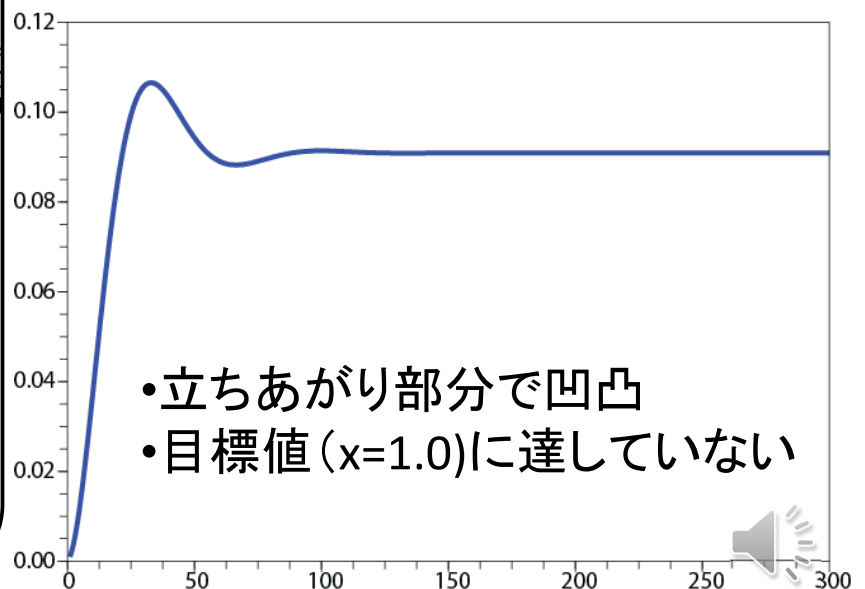


Scilabコード

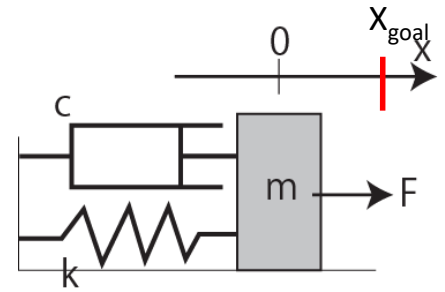
```
m=1.0; //質量
c=1.0; //ダンパ
k=1.0; //バネ
xgoal=1.0; //目標値
x=0; //現在地
v=0; //現在速度
dt = 0.1; //時間間隔
xrecord = []; //記録用
```

```
for t=1:300,
  F=0.1 * (xgoal-x);
  a = (F - k*x - c*v)/m;
  v = v+a*dt;
  x = x+v*dt;
  xrecord = [xrecord,x];
end
```

```
plot(xrecord);
```



# ダンパが大きかったら？

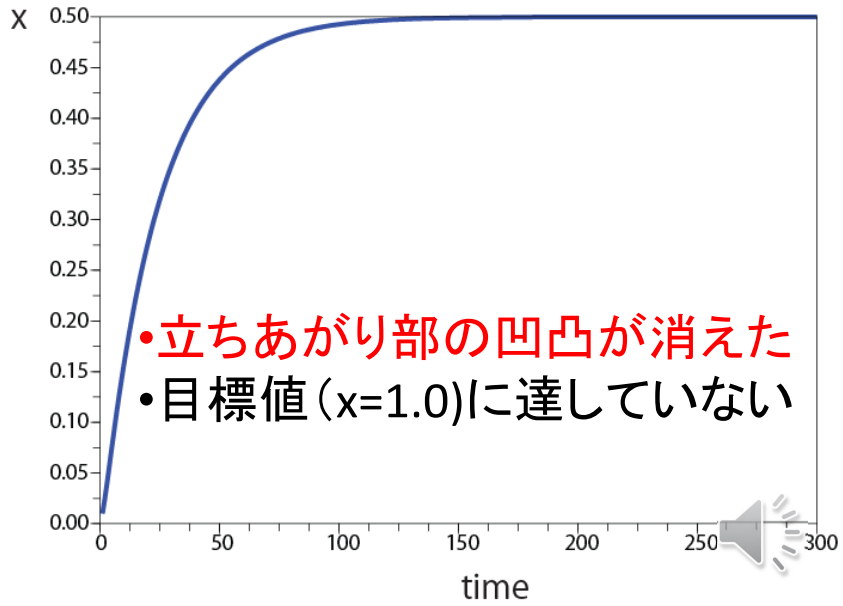


```

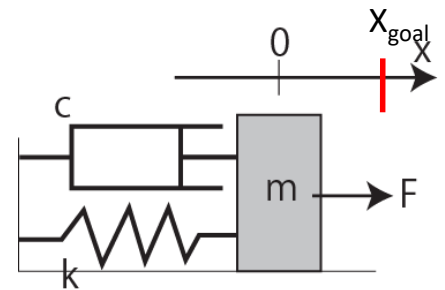
Scilabコード
m=1.0;           //質量
c=5.0;         //ダンパ
k=1.0;         //バネ
xgoal=1.0;     //目標値
x=0;          //現在地
v=0;         //現在速度
dt = 0.1;     //時間間隔
xrecord = [];  //記録用

for t=1:300,
    F=1.0 * (xgoal-x);
    a = (F - k*x - c*v)/m;
    v = v+a*dt;
    x = x+v*dt;
    xrecord = [xrecord,x];
end

plot(xrecord);
    
```

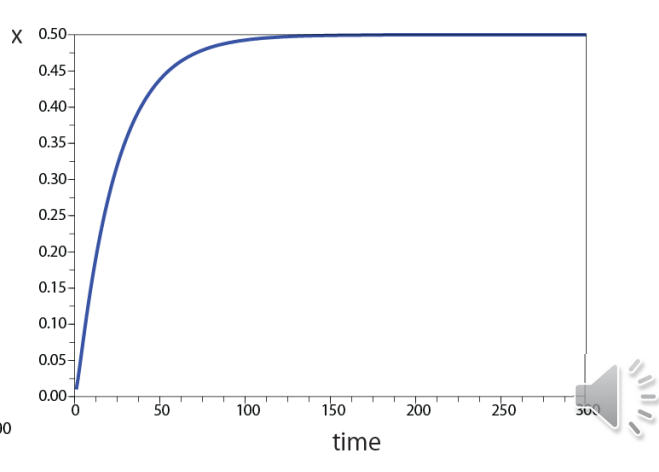
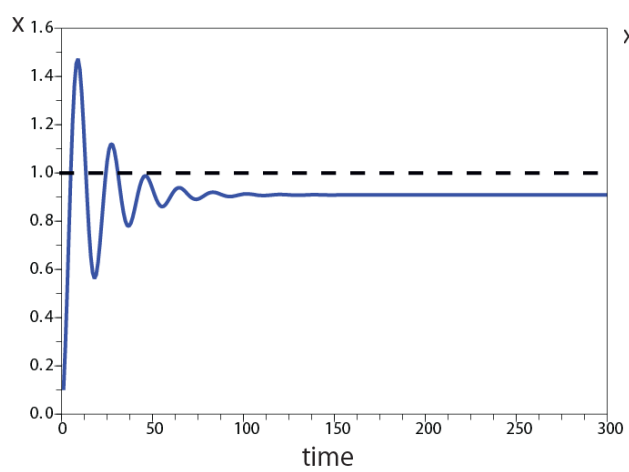


# P制御のまとめ

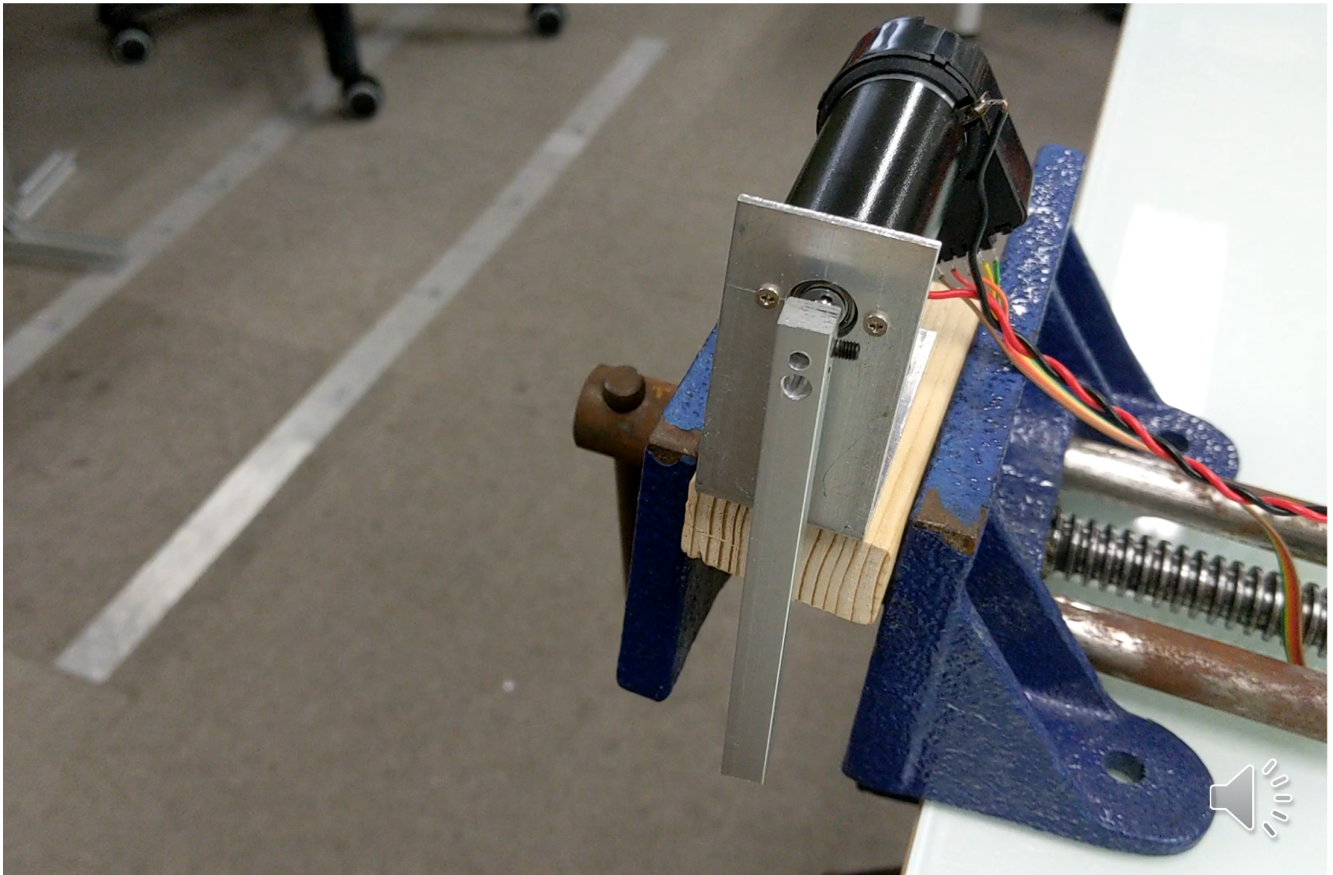


どうやら...

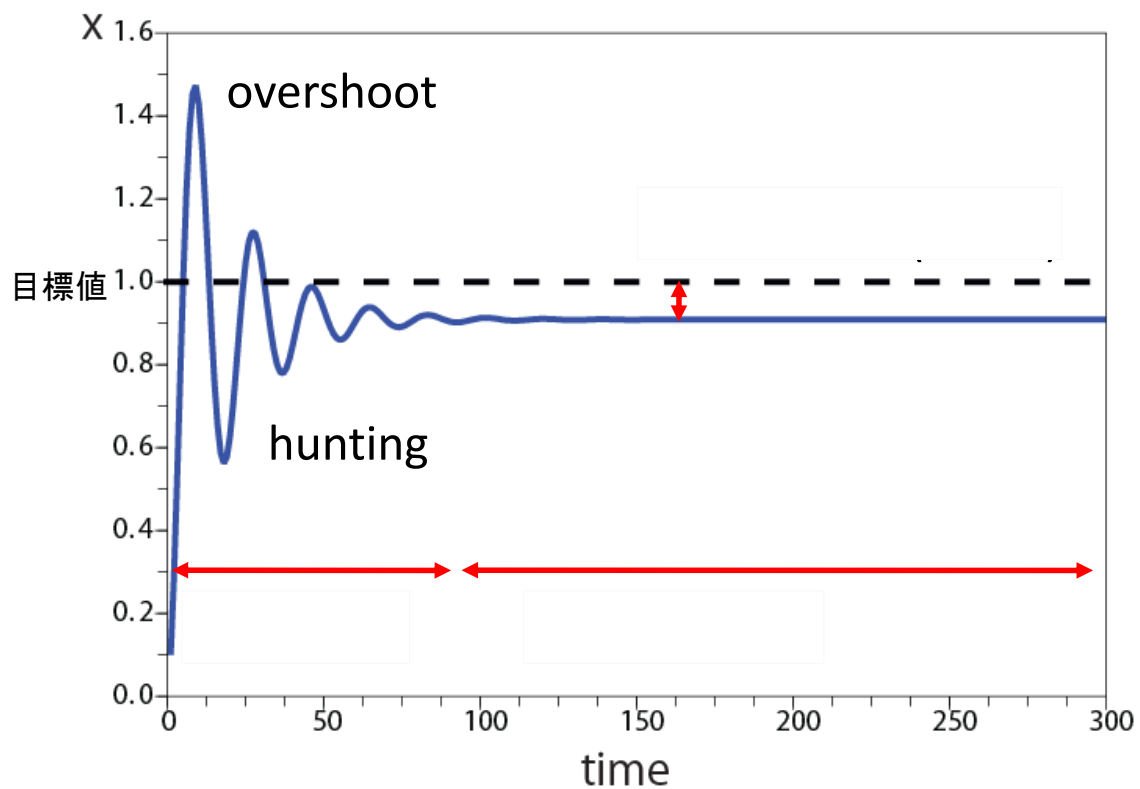
- 最終的に目標値 (x=1.0) に達しない？
- はじめに振動するかどうかは、ダンパの大きさに依存？



# モータのP制御



## 用語



# P制御の数学

- システム

$$m\ddot{x} + c\dot{x} + kx = f$$

- 力の制御の仕方

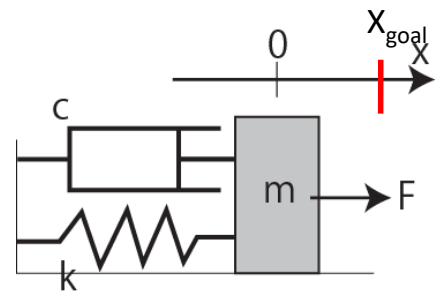
$$f =$$

- つまり

$$m\ddot{x} + c\dot{x} + kx =$$

- 一般化して次の式を考えよう

$$\ddot{x} + a\dot{x} + bx = c$$



# P制御の数学

$$\ddot{x} + a\dot{x} + bx = c$$

- ラプラス変換すると

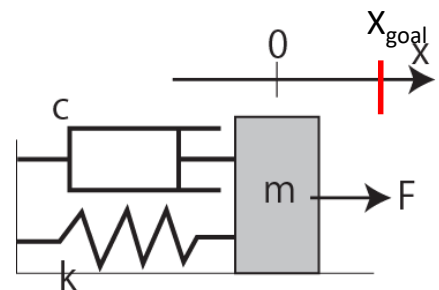
$$s^2 X(s) + a s X(s) + b X(s) = \frac{c}{s}$$

- つまり、軌道  $x(t)$  のラプラス変換  $X(s)$  は、

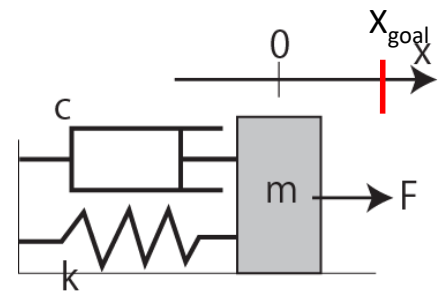
$$X(s) = \frac{c}{s(s^2 + a s + b)}$$

- 2次方程式の根を  $\lambda_1, \lambda_2$  とすると、

$$X(s) = \frac{A}{s} + \frac{B}{s - \lambda_1} + \frac{C}{s - \lambda_2}$$



# P制御の数学



- 2次方程式の根を $\lambda_1, \lambda_2$ とすると,

- 部分分数分解をすると,

- 結局, 逆ラプラス変換をすると

つまり,

- 二次方程式の根 $\lambda_1, \lambda_2$ が過渡的なふるまいを決定し
- 定数項 $a_1$ が, 収束値を決定する.



## P制御の数学 (2次方程式の根)

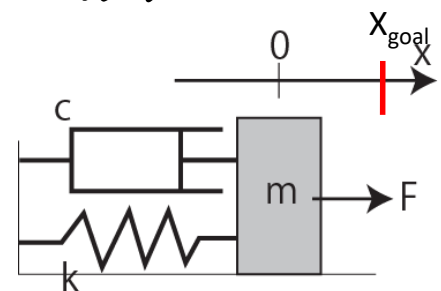
- 2次方程式の根 $\lambda_1, \lambda_2$ について

$$m\ddot{x} + c\dot{x} + kx = a(x_{goal} - x)$$

- 根 $\lambda_1, \lambda_2$ は,  $ms^2 + cs + (k + a) = 0$ の根.

$$\lambda = \frac{-c \pm \sqrt{c^2 - 4m(k + a)}}{2m}$$

- 高校生でもわかることが2つ!
- (1) $\lambda$ の実部は負である
- (2) $c^2$ が $4m(k+a)$ よりも小さいと,  $\lambda$ は虚部を持つ



## P制御の数学（2次方程式の根）

$$\lambda = \frac{-c \pm \sqrt{c^2 - 4m(k+a)}}{2m}$$

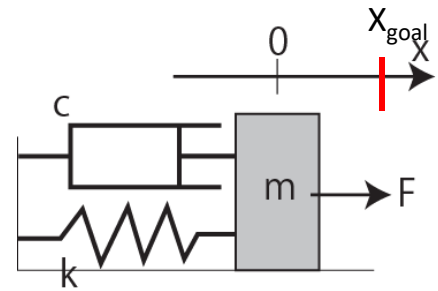
(1)  $\lambda$ の**実部は負**である

だから,

$$x(t) = a_1 + a_2 \exp(\lambda_1 t) + a_3 \exp(\lambda_2 t)$$

のexpの項はすぐに減衰する.

つまり, **無限大に発散することはない(ひと安心!)**



## P制御の数学（2次方程式の根）

$$\lambda = \frac{-c \pm \sqrt{c^2 - 4m(k+a)}}{2m}$$

(2)  $c^2$ が $4m(k+a)$ よりも小さいと,  $\lambda$ は**虚部を持つ**

このとき,

$$x(t) = a_1 + a_2 \exp(\lambda_1 t) + a_3 \exp(\lambda_2 t)$$

の, expの項は,  $\exp(-c_1 t + ic_2 t) = \exp(-c_1 t) \cdot \exp(ic_2 t)$

つまり,

- **減衰**する成分  $\exp(-c_1 t)$  と,
- **振動**する成分  $\exp(ic_2 t) = \cos(c_2 t) + i \sin(c_2 t)$

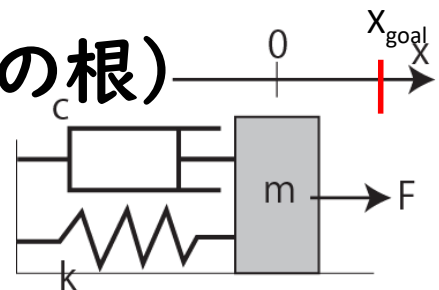
に分けられる.

これこそが, はじめの振動の原因



# P制御の数学（2次方程式の根）

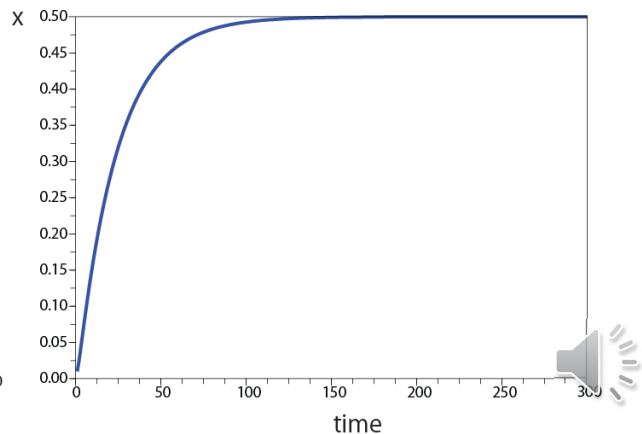
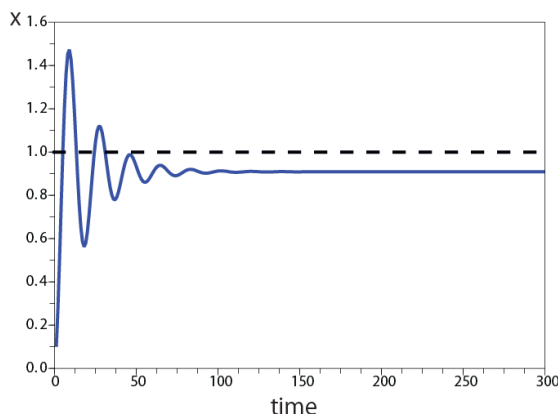
$$\lambda = \frac{-c \pm \sqrt{c^2 - 4m(k+a)}}{2m}$$



(2)  $c^2$ が $4m(k+a)$ よりも小さいと,  $\lambda$ は**虚部を持つ**

これこそが, はじめの振動の原因

つまり, 「振動を抑えるためにはダンパ(ブレーキ)を大きくすればよい」ということ.



# P制御の数学（再掲）

• 2次方程式の根を $\lambda_1, \lambda_2$ とすると,

$$X = \frac{c}{s(s - \lambda_1)(s - \lambda_2)}$$

• 部分分数分解をすると,

$$X = \frac{a_1}{s} + \frac{a_2}{(s - \lambda_1)} + \frac{a_3}{(s - \lambda_2)}$$

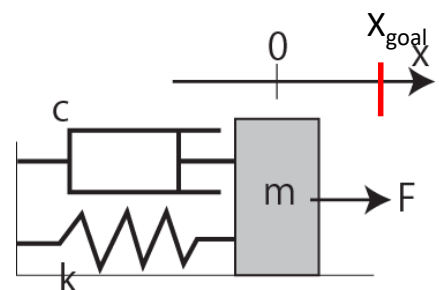
• 結局, 逆ラプラス変換をすると

$$x(t) = a_1 + a_2 \exp(\lambda_1 t) + a_3 \exp(\lambda_2 t)$$

• つまり,

• 二次方程式の根 $\lambda_1, \lambda_2$ が過渡的なふるまいを決定し

• 定数項 $a_1$ が, 収束値を決定する.





# P制御の数学（定数項）

$$X = \frac{c}{s(s - \lambda_1)(s - \lambda_2)}$$

$$X = \frac{a_1}{s} + \frac{a_2}{(s - \lambda_1)} + \frac{a_3}{(s - \lambda_2)}$$

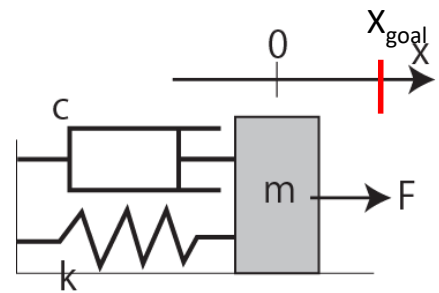
- 結局, 逆ラプラス変換をすると

$$x(t) = a_1 + a_2 \exp(\lambda_1 t) + a_3 \exp(\lambda_2 t)$$

- a1**以外は時間がたてば消えるので, **a1**が**収束値**となる。

a1は部分分数分解を頑張らなければ求められない？

**NO !**



# P制御の数学（定数項）

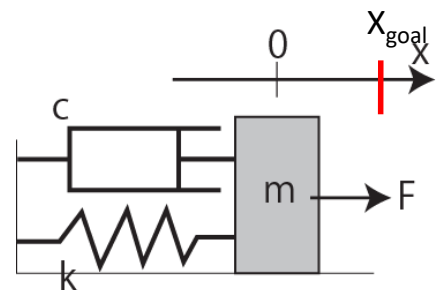
- システムの応答の式:

$$m\ddot{x} + c\dot{x} + kx = a(x_{goal} - x)$$

- いま考えたいのは「定常的」になった時だから,

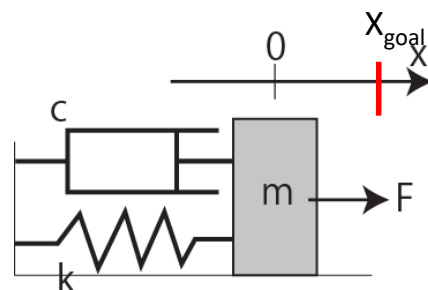
よって

すなわち



# P制御の数学（定数項）

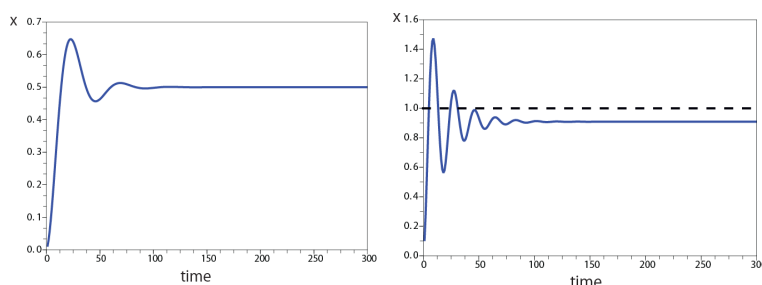
収束値: 
$$x = \frac{a}{k+a} x_{goal}$$



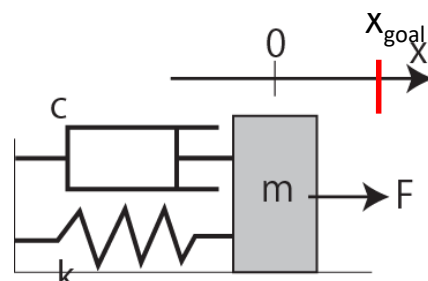
各定数の意味は, k:ばね定数, a:P制御の比例定数

よって次のことがわかる

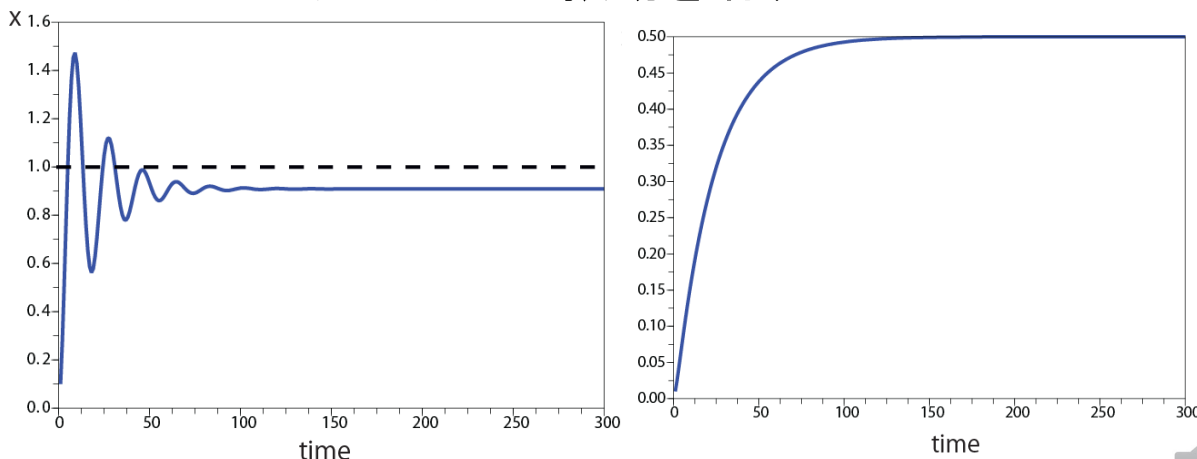
- (1) aを大きくすれば, 収束値は目標値に近づく
- (2) しかし, 収束値は目標値より常に小さい
- (3) ただし「ばね成分」が0ならちゃんと収束する.



# P制御のまとめ（再掲）



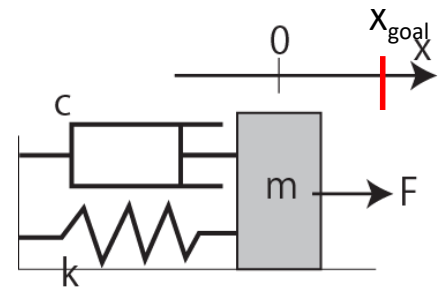
- 最終的に目標値 (x=1.0) には**達しない**
- ただし, 比例ゲインが大きければ目標値に近づく
- はじめに**振動**するかどうかは, **ダンパの大きさ**に依存.
- ダンパが大きければ振動を消すことができる



ラプラス変換によって, 数学的に理解できた



# レポート課題(1)



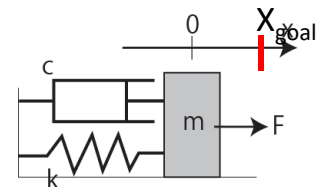
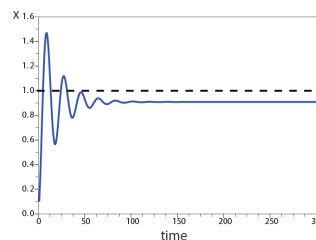
- P制御で振動しないためのダンパ $c$ の大きさの数値的な条件を求めよ。  
ただし,  $k=1$ ,  $m=1$ とする。

- ダンパをその値周辺(例えば $\pm 0.5$ )に設定したシミュレーションを行い, 実際に振動が抑えられることを確認せよ。

- すべてコード中に記載すること。



## PI制御



- P(比例)制御は, バネの影響で目標値に収束しない

- これを改善するため, 「積分(Integral)」を用意する。

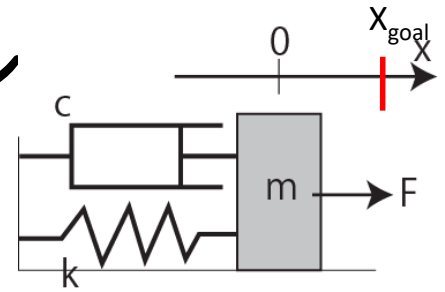
- PI制御:

- 目標値との誤差(P)成分と,
- その誤差成分の時間的な累積(I)とを,  
適当な係数で足し合わせて制御信号とする。



# PI制御のシミュレーション

P成分: 目標位置と現在位置の誤差  
I成分: 誤差の積分(累積)



Scilabコード

```
m=1.0; //質量
c=5.0; //ダンパ
k=1.0; //バネ
xgoal=1.0; //目標値
x=0; //現在地
v=0; //現在速度
dt = 0.1; //時間間隔
xrecord = []; //データ記録用
P=0.8; //P成分
I=0.05; //I成分
i_seibun = 0;
```

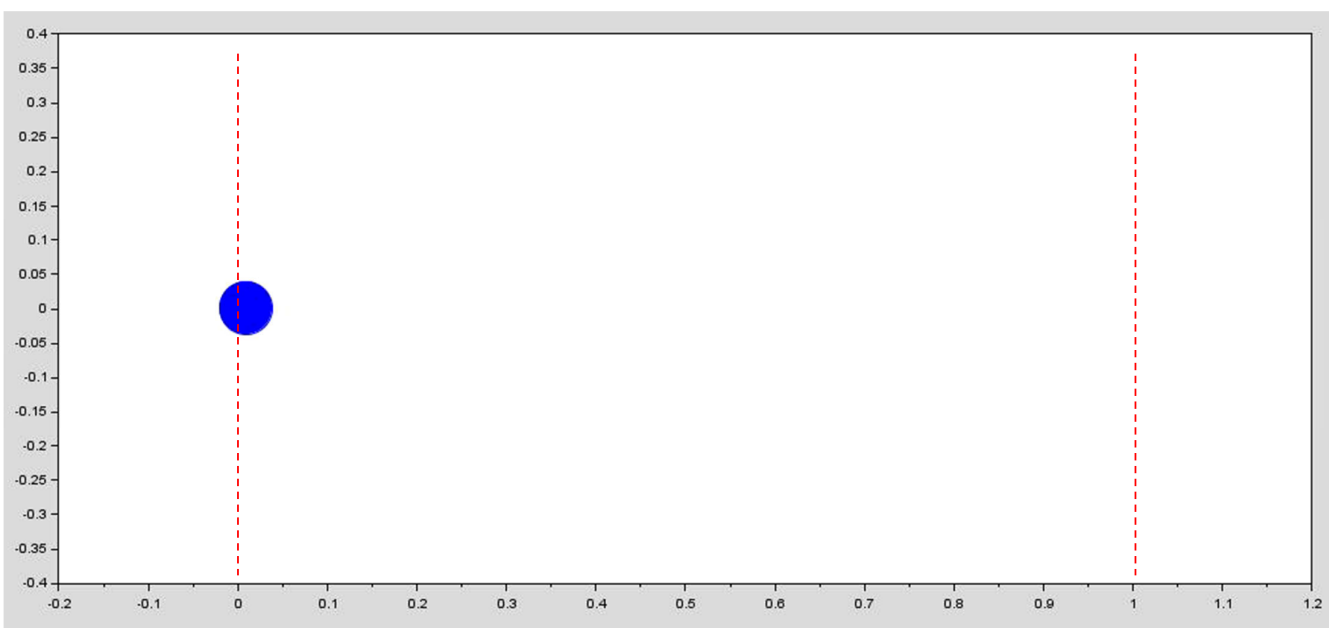
```
for t=1:300,
  p_seibun =
  i_seibun =
  F=
  a = (F - k*x - c*v)/m;
  v = v+a*dt;
  x = x+v*dt;
  xrecord = [xrecord,x];
end

plot(xrecord);
```

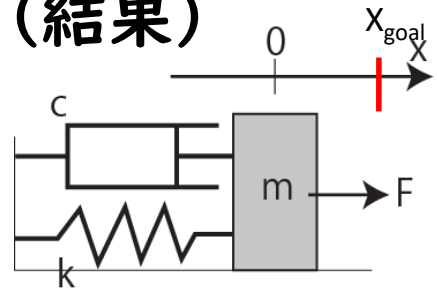
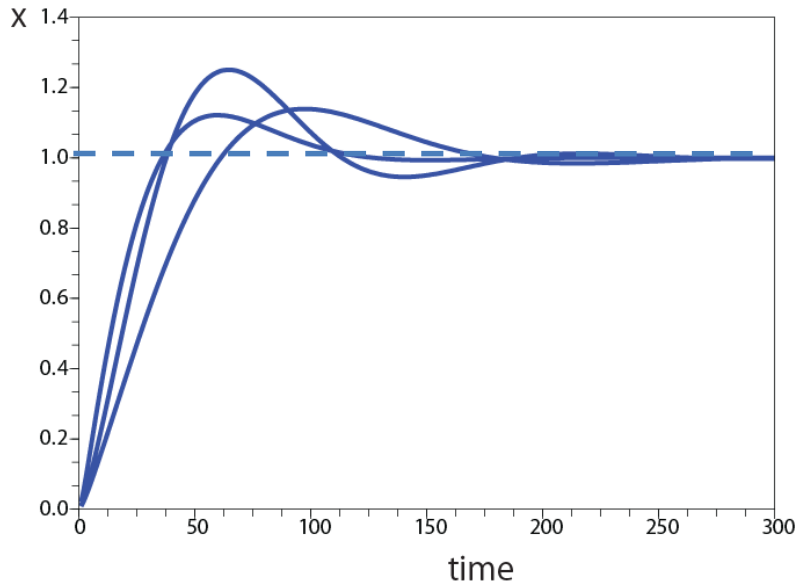
I成分のイメージ:  
誤差があると, **どんどん累積**して無視できなくなる



## PI制御



# PI制御のシミュレーション (結果)



P成分とI成分を色々変えてみた.

- 確かに、収束値は目標値と一致する.
- ただし目標値を行きすぎる(振動成分を持つ)こともあるようだ.



# PI制御の数学 (最終状態について)

- システム

$$m\ddot{x} + c\dot{x} + kx = f$$

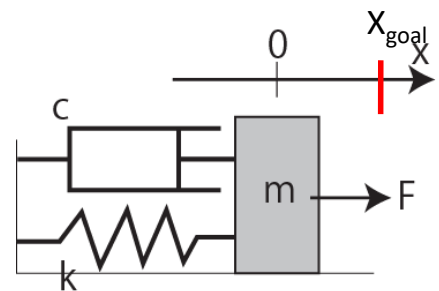
- 制御方式

- つまり

- 両辺を微分して

- 時間が無限に経過した定常状態では

- よって、最終的に目標値に一致する.



# PI制御

- 最終的には目標値に一致する.
- 比例ゲインが大きいと振動する.
- 比例ゲインが小さいと収束は遅い.

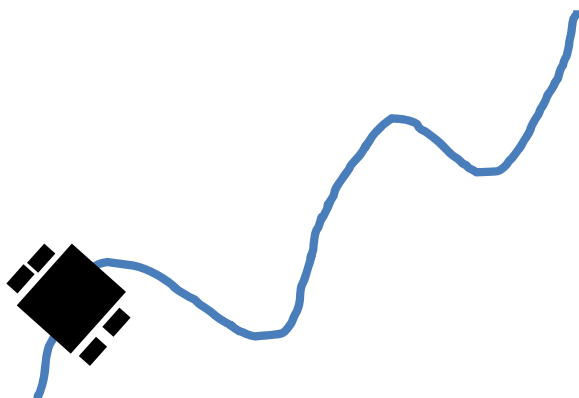
•ゆっくりでもよいから完全に目標値に合わせたいときに使う.

- (例)温度制御
  - エアコン
  - 化学プラント

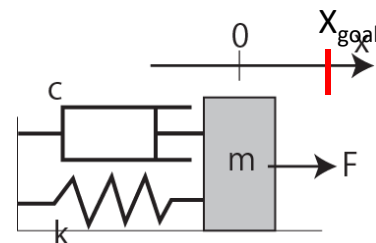


# PI制御のデメリット

- Iは積分, すなわち**時間遅れを含む**.
- ある一定の目標に達するのが目標ならOK.  
だが, 目標値が時々刻々と変化する(軌道に沿って動かす等)場合, I成分による時間遅れが問題となる.

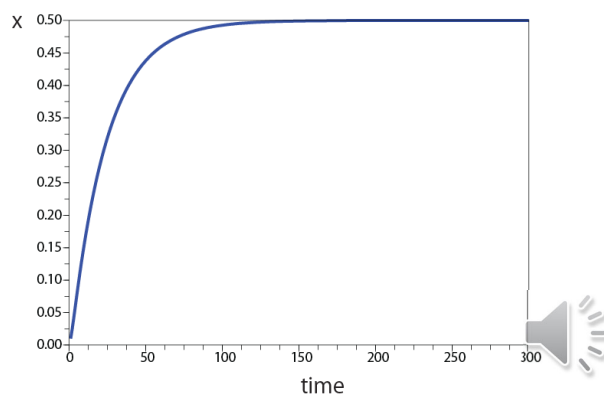
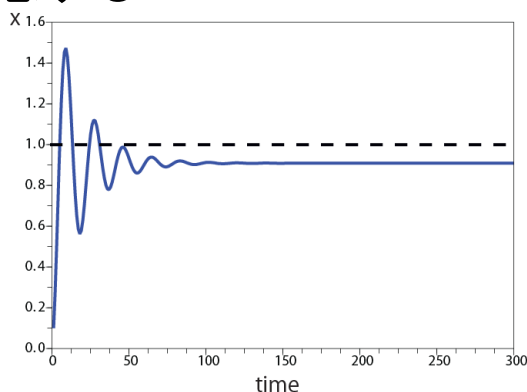


# PD制御



- P(比例)制御では,  
振動を抑えるために大きなダンパを用意する必要.
- 言いかえれば, システムを物理的に変える必要.

•これを改善するため, バーチャルなダンパ(ブレーキ)を用意する.



## PD制御のシミュレーション

- P成分: 目標値と現在地の差に比例
- D成分: 速度に比例したブレーキ. つまりダンパ

Scilabコード

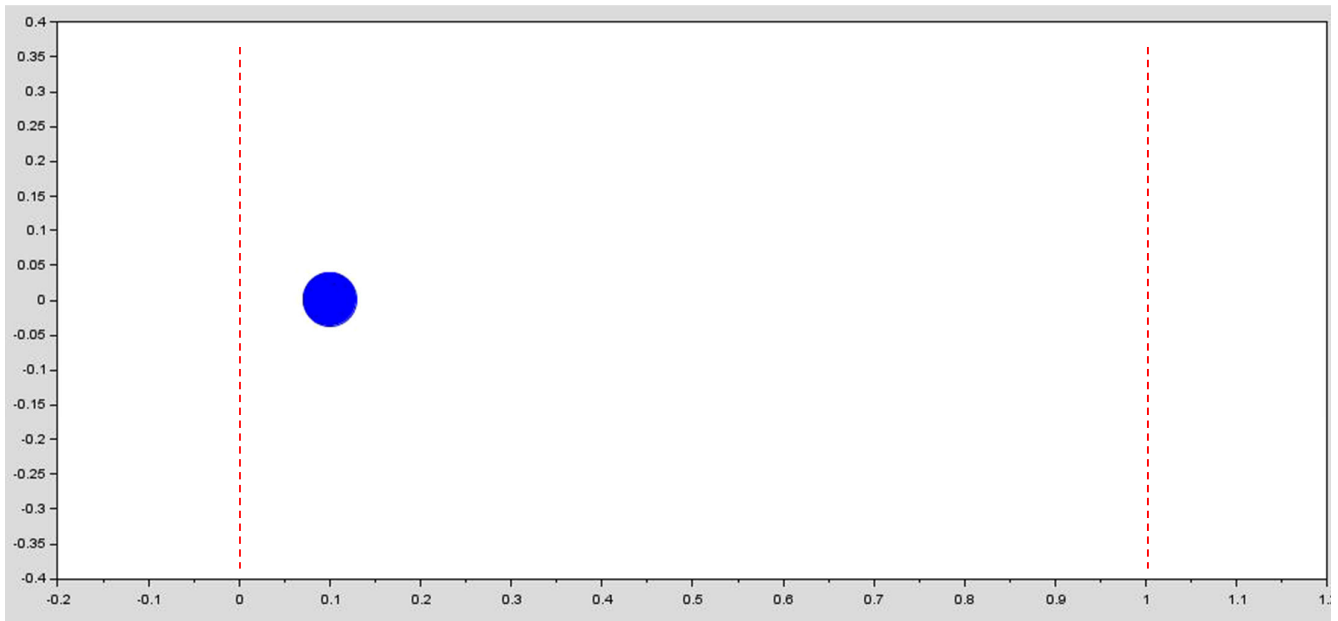
```
m=1.0;           //質量
c=1.0;           //ダンパ
k=1.0;           //バネ
xgoal=1.0;       //目的地
x=0;             //現在地
v=0;             //現在速度
dt = 0.1;        //時間間隔
xrecord = [];    //データ記録用
P=1.0;           //P成分
D=1.0;           //I成分
```

```
for t=1:300,
    F=
    a = (F - k*x - c*v)/m;
    v = v+a*dt;
    x = x+v*dt;
    xrecord = [xrecord,x];
end

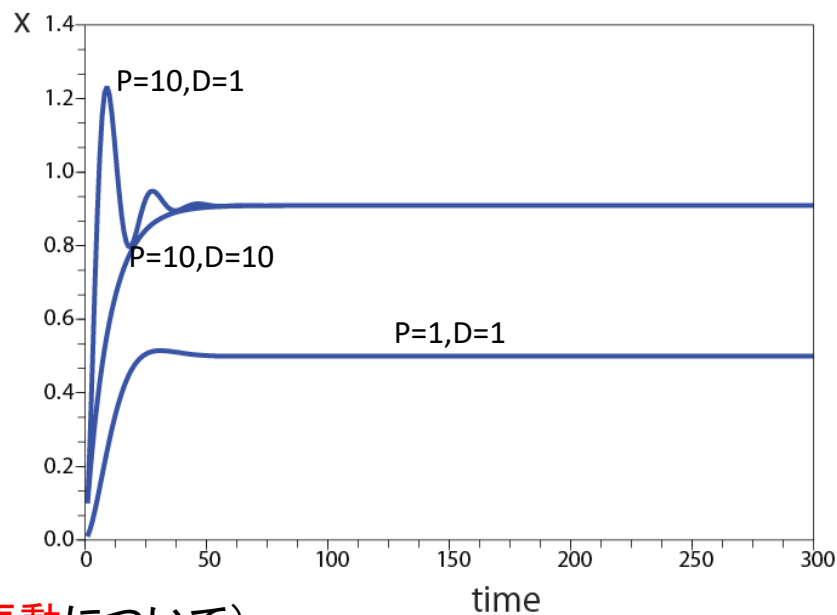
plot(xrecord);
```



# PD制御 : $P=10, D=1$



## PD制御のシミュレーション(結果)

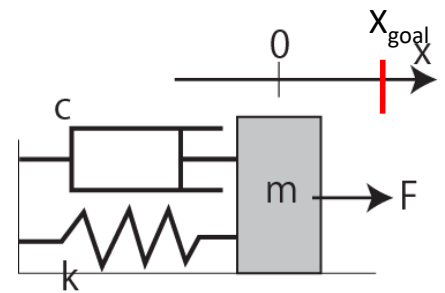


- (振動について)
  - Dを大きくすれば振動が抑えられる.
  - でもPを大きくしたらやっぱり振動する.
  - それでもDをもっと大きくすれば振動は抑えられる.
- (収束について)
  - P制御と同様に定常偏差がある(目標値に収束しない)





# PD制御の数学



## •システム

$$m\ddot{x} + c\dot{x} + kx = f$$

## •力の制御の仕方

## •つまり

•これは、P制御において、ダンパ成分がcからb+cに増えたことを意味する。



# PD制御の数学

•これは、P制御において、ダンパ成分がcからb+cに増えたことを意味する。

$$m\ddot{x} + (b + c)\dot{x} + kx = a(x_{goal} - x)$$

•P制御で振動しない条件は： $\lambda = \frac{-c \pm \sqrt{c^2 - 4m(k+a)}}{2m}$  が虚部を持たないことだった。つまり、

$$c^2 - 4m(k + a) \geq 0$$

•これが、PD制御では、ダンパ成分が増えたことにより、

$$(b + c)^2 - 4m(k + a) \geq 0$$

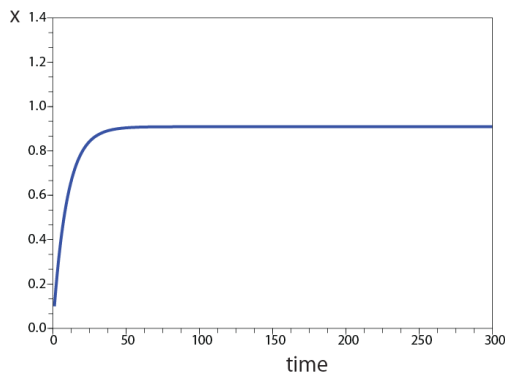
•つまり、より振動しにくくなった。

•P制御と同じだから定常偏差が残る。

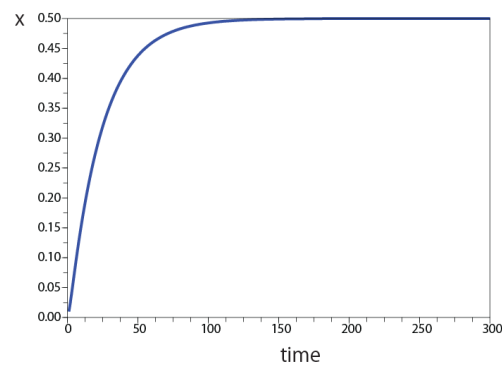


# PD制御の利点

- ダンパが等価的に増えることで、「振動しにくくなる」
  - だからPゲインを思い切り上げられる
  - だから結果としてP制御に比べて目標到達速度が速い
- 厳密に目標値に達することよりも、高速に移動することが目標の場合に多く使われる。



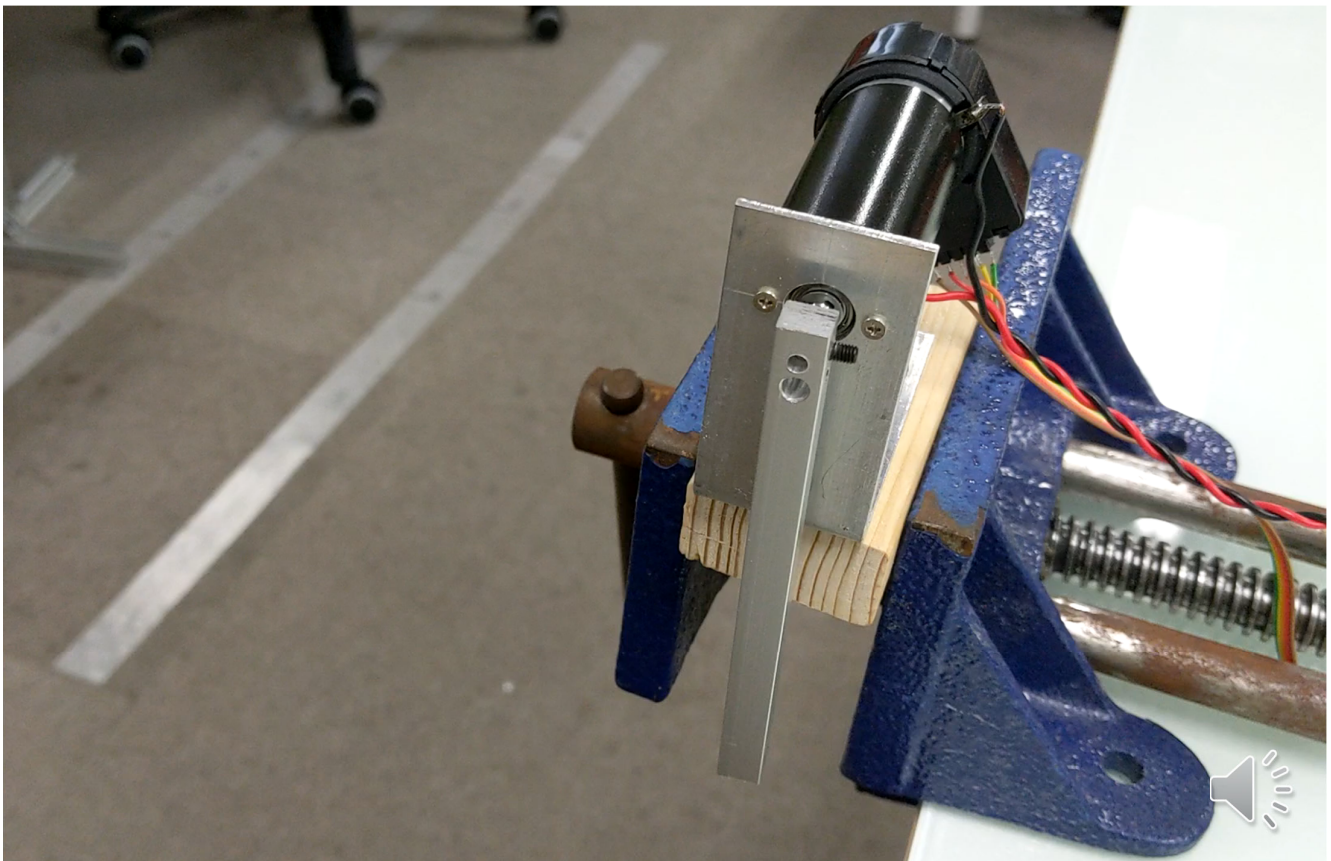
PD制御 : P=10, D=10



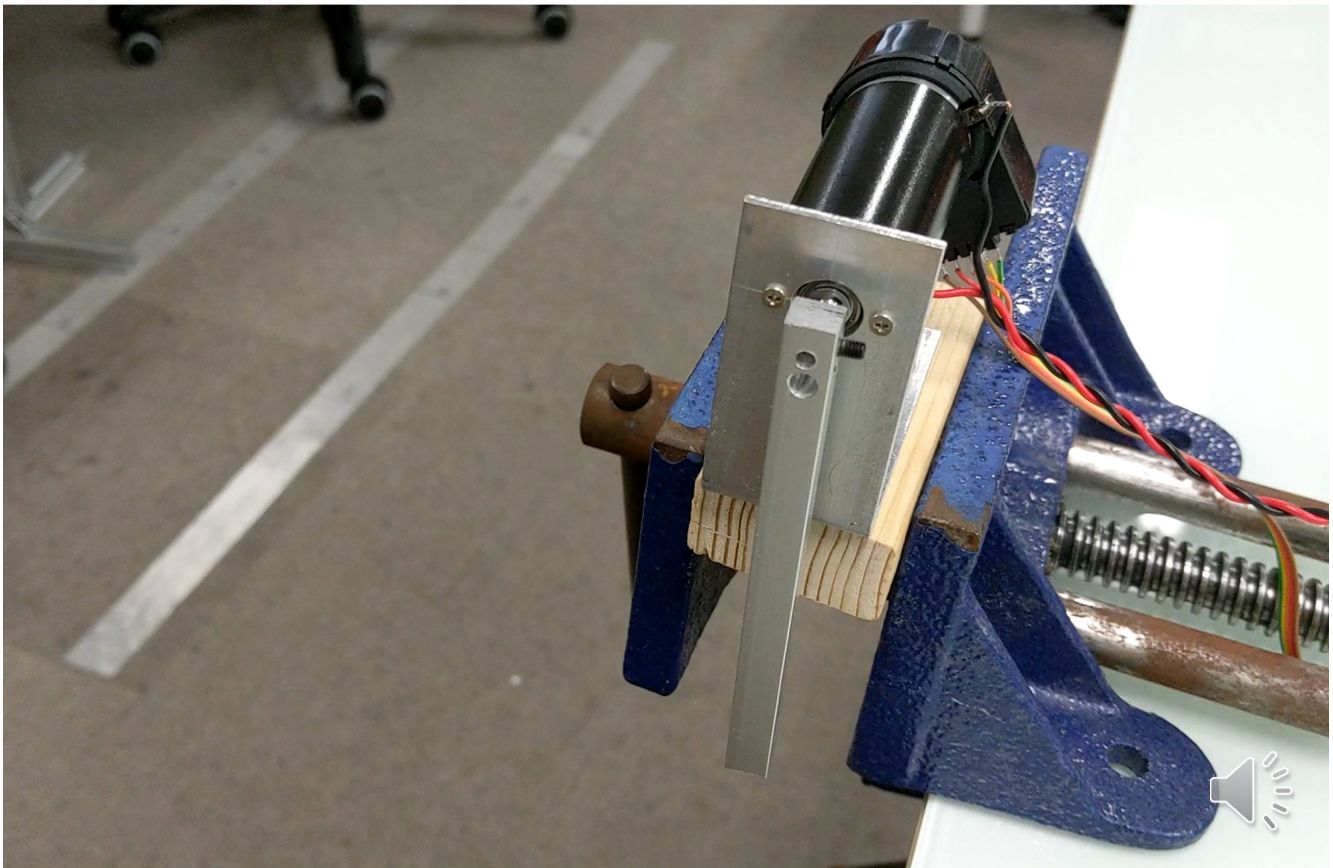
P制御 : P=0.5



# モータのPD制御(1)



## モータのPD制御(2)



## モータとPD制御

(再考) 収束値:  $x = \frac{a}{k+a} x_{goal}$



- モータには通常「ばね成分」はない.
- よって収束値が目標値とずれるという問題が生じにくい
- このため、「目標値が時々刻々と変化する」(＝軌道に沿って動かす)場合、PD制御が多く使われる.
- ただし重力がある場合、やはり目標値とずれる問題が生じる。重力補償項を設けることも。

# PID制御

ここまでのすべてを合わせたもの.

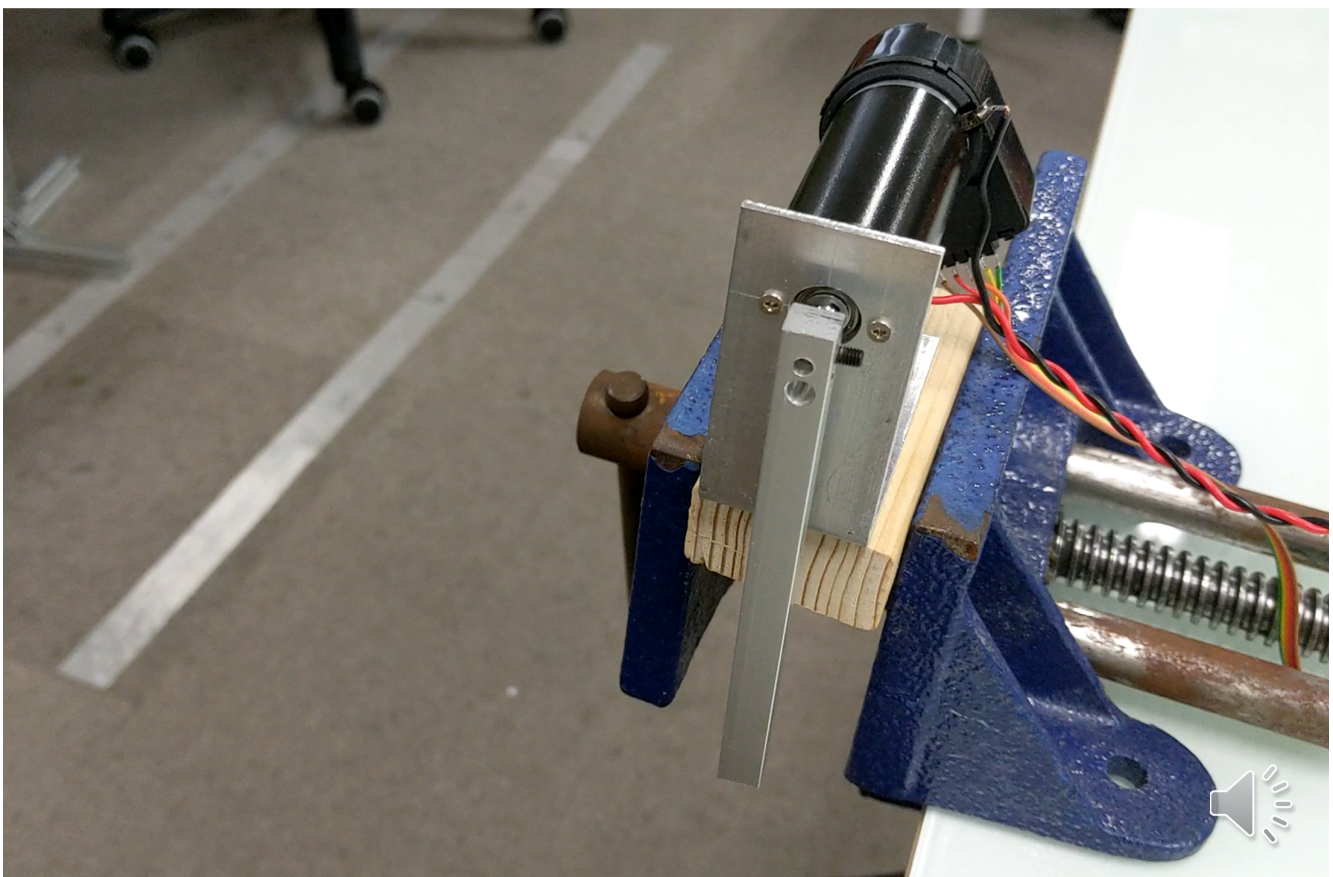
- P:現在の値と目標値との誤差(比例成分=Proportional)
- I:誤差の積分(Integral)
- D:速度(微分成分=Derivative)

それぞれの役割は

- P:**早く**目標に達する.
- I:**定常偏差**を無くす
- D:**振動**を抑える.



## モータのPID制御



# (再考) 伝達関数と制御

$$m\ddot{x} + c\dot{x} + kx = f$$

両辺をラプラス変換すると

$$(ms^2 + cs + k)X = F$$

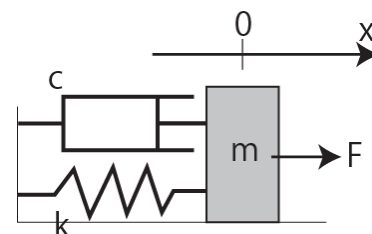
$$X(s) = \frac{F}{ms^2 + cs + k}$$

つまり、制御とは、元のシステム

$$G(s) = \frac{1}{ms^2 + cs + k} \quad (\text{伝達関数})$$

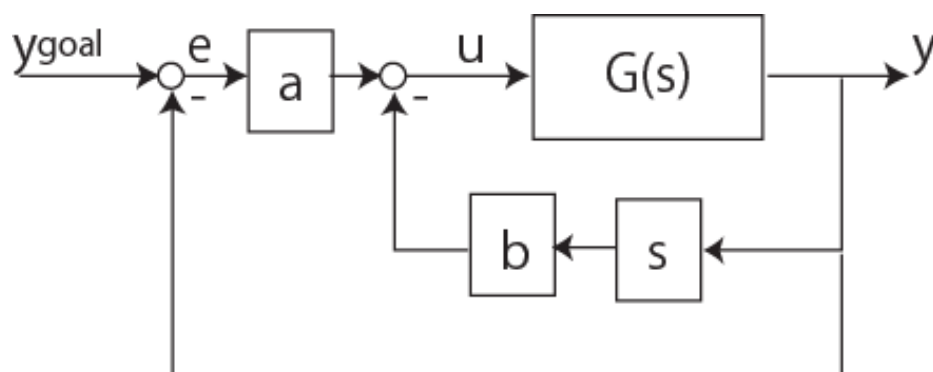
に、適切な入力  $F(s)$  を与えて、

望ましい軌道  $X(s)$  を得る操作である。



## ブロック線図

制御の流れを図にしたもの。微分をs, 積分を1/sと書く。



ygoal: 目標値, y: 現在の値, e: 誤差, u: システムへの入力  
a: 比例ゲイン, b: 微分ゲイン, G: システムの伝達関数

流れに従って考えれば、自然にラプラス変換表現が得られる。  
上図はPD制御の場合。



## レポート課題 (2)

これまでと同じバネマスダンパ系に対してPID制御を実装したうえで、P,I,Dの係数を変化させ、

- (1)なるべく早く目標値に達し、
- (2)振動しない(オーバーシュートがない)ようにせよ。

※実は大きければ大きいほどよくなる。実際の制御では、出力の制限が制御性能をほとんど決めてしまう。

※余裕があれば適当に出力の最大値の制限を設けたうえで試してみよ。



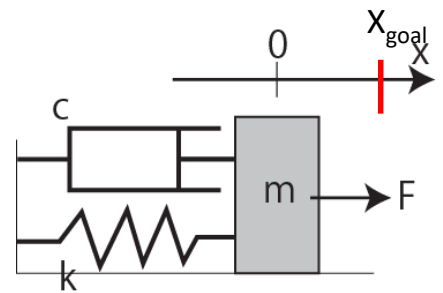
# インピーダンス制御



# 一般化：インピーダンス制御

- システム(バネ成分は無いことも多い)

$$m\ddot{x} + c\dot{x} + kx = f$$



- 力の制御の仕方

- つまり

- これは、元のインピーダンス $m, c, k$ を、 $m+m', c+c', k+k'$ に変化させたことを意味する。



## 力覚ディスプレイ=インピーダンス提示装置

