



Óbudai egyetem

Bánki Donát

Gépész és Biztonságtechnikai Mérnöki Kar

Mechatronikai Mesterképzés

Adaptív szabályozás

Adaptív Self tuning szabályozás

Tanár Dr. Sárceivity Péter

Tanuló: Kovács Árpád

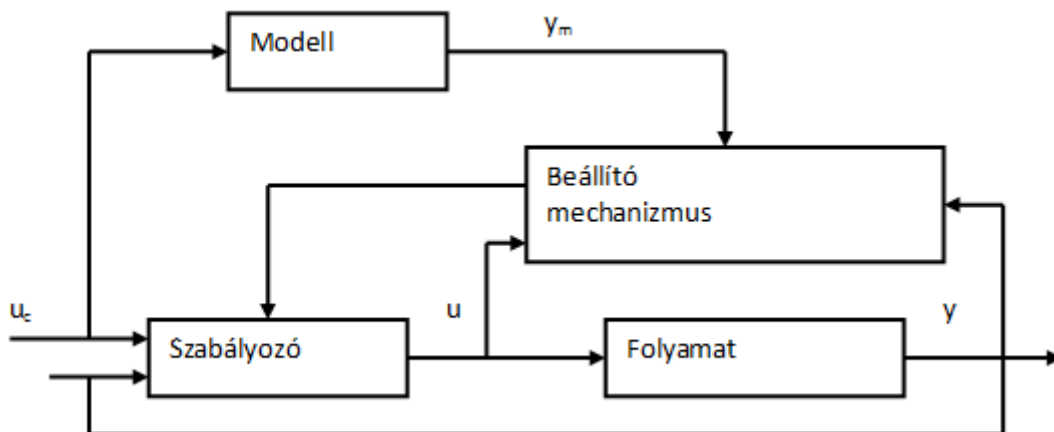
Neptun Kód: BPJZ56

Adaptív szabályozás

Általában a tervezési cél az, hogy olyan szabályozót válasszunk, amely lehetővé teszi, hogy a szabályozott szakasz - azaz a folyamat - kimenete kövesse a bemeneti referenciajel által előírt jelet. Az általános bemenőjelen és beavatkozó jelen kívül vannak speciális bemenőjelek is, a zavarok. Ezeket nem mi szabályozzuk, mégis befolyásolják a kimenet viselkedését. Ezért a szabályozót úgy kell megtervezni - az előzőek figyelembevételével -, hogy a zavaró hatások a szabályozott szakasz kimenetén már kompenzálódjanak. Tehát a két fő tervezési cél: a pályakövetés és a zavarkompenzáció megvalósítása. Ez minden szabályozási módszernek a központi kérdése [6].

A szabályozásnak eleget kell tennie a szabályozni kívánt műszaki folyamattal szemben támasztott minőségi követelményeknek. A minőségi követelmények egyrészt a szabályozás statikus pontosságát, azaz állandósult hibáját, másrészt dinamikus viselkedését, azaz szabályozási idejét, túl lendülését írják elő. A valóságos és az előírt minőségi jellemzők összehasonlítása a szabályozási rendszer dinamikus viselkedésének vizsgálata alapján végezhető el.

A minőségi követelményeket egy modell segítségével is megfogalmazható, vagyis egy modell segítségével adható meg, hogy különböző alapjelre milyen válasz kívánatos a visszacsatolt rendszertől.



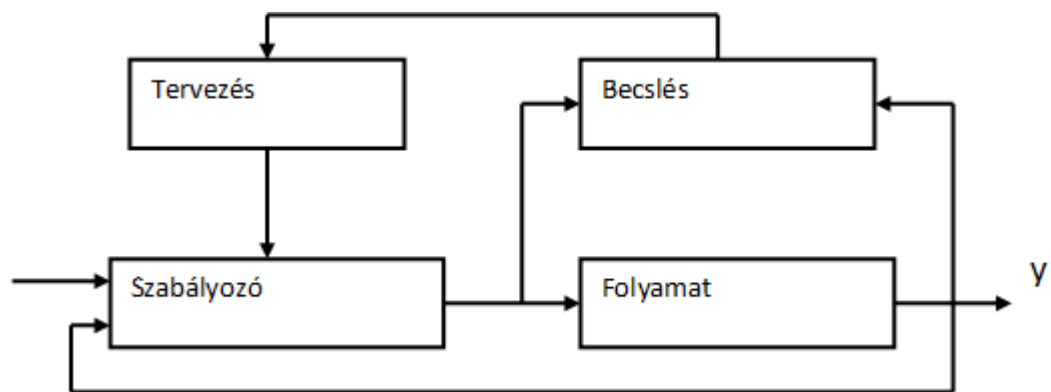
Ha a szabályozott szakasz paraméterei állandóak és azokat pontosan ismerjük, akkor pontosan meghatározható, hogy milyen kompenzáló tagra van szükség a megkívánt viselkedés eléréséhez (feltéve, hogy nem támasztunk megvalósíthatatlan követelményeket a rendszerrel szemben). A gondolatsort folytatva eljutunk a modell referenciás adaptív szabályozáshoz (a későbbiekben részletesen tárgyaljuk). A modell referenciás szabályozóknál létezik egy szokásos visszacsatolt kört egy kompenzáló taggal. A visszacsatolt szakasz viselkedése egy referencia modell segítségével definiálható. A modell és a visszacsatolt szakasz ugyanazt az alapjelet kapja meg.

Az adaptív rendszerek nemlineáris. A viselkedésük egészen összetett és ez igen nehézvé teszi az analízisüket. Az elmélet lassan fejlődött és sok munkába került, míg egy meglehetősen teljes és összefüggő elmélet jött létre. Természetesen ez az elmélet azért ma sem lezárt. Az adaptív rendszereket komplex viselkedésük következtében különböző szempontokból szükséges figyelembe venni. A nemlineáris rendszerek elmélete, a stabilitás, a rendszer identifikáció, a rekurzív paraméterbecslés, az optimális szabályozás és a sztochasztikus szabályozás mind hozzájárulnak az adaptív rendszerek megértéséhez.

Az adaptív rendszerek elméletének sokféle célja van. Elvárható, sőt szükséges, hogy olyan eszközök álljanak a rendelkezésünkre, amelyekkel egy adott rendszert tervezési szempontból elemezni tudunk. Egy tipikusnak tekinthető problémamegfogalmazás: olyan paraméter beállítási szabályt kell alkotni, amely gondoskodik a kívánt eredményekről egy stabil, zárthurkú rendszerben.

Self Tuning szabályozás

Ebben a sémában a szabályozott szakasz paramétereit folyamatosan megbecsüljük, a becsült paraméterek ismeretében folyamatosan újra tervezzük kompenzáló tag paramétereinek értékét. A self-tuning adaptív szabályozás alapstruktúrája itt látható.



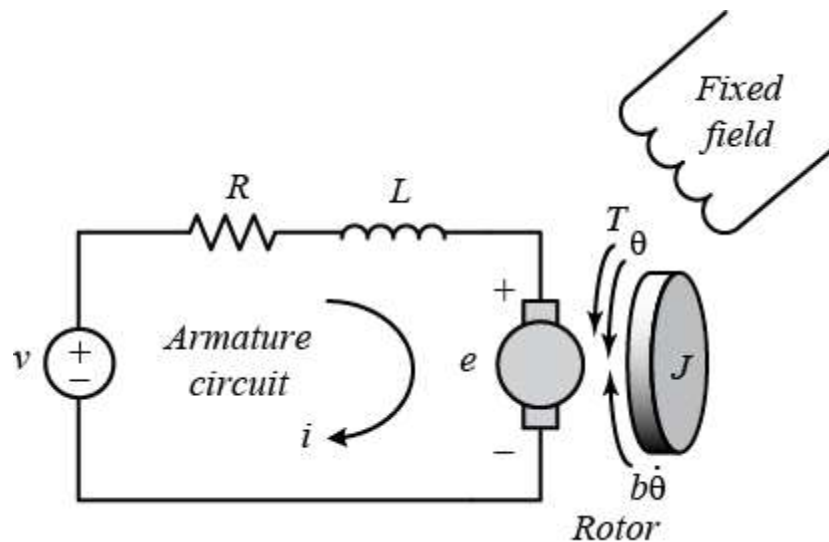
A self-tuning szabályozási rendszer két hurkot tartalmaz. A belső hurok a folyamatból és egy közös lineáris visszacsatolásból áll. A szabályozási paraméterek a külső hurok által szabályozzuk, amely egy rekurzív paraméterbecslőből és egy ún. tervezőből áll. Sokféle self-tuning szabályozási sémát kaphatunk a különböző tervezési és becslési algoritmusok kombinálásával. A szabályozás további fontos tulajdonságai és példákra való alkalmazásának bemutatása a irodalomban található.

Harmadikként a sztochasztikus adaptív szabályozásról szólunk tömören. A self-tuning szabályozásnál a kompenzáló tag paramétereinek tervezésekor a szabályozott szakasz becsült paramétereit már pontosnak tekintjük, nem vesszük számításba a becslés esetleges pontatlanságát. A sztochasztikus adaptív szabályozás ebben a tekintetben jelent továbblépést, a szabályozott szakasz paramétereit nem egy konkrét értékkel adjuk meg, hanem leképezzük a valószínűségi térbe, és itt vizsgáljuk a valószínűségi eloszlásukat, és ennek megfelelően avatkozhatunk be a szabályozási körbe. A sztochasztikus adaptív szabályozás részletes bemutatása a irodalomban található.

DC motor pozicionálása

Fizikai felépítés:

A vezérlő rendszerekben a leggyakrabban használt végrehajtó szerv a DC motor. Forgó mozgást végez, amit kerekek vagy egyéb eszközök segítségével tranlációs mozgássá tudunk alakítani. A motor felépítése a következő ábrán látható (1. ábra).



A példa bemutatásához a motor következő fizikai paramétereit vesszük fel:display(n)

J a forgó rész tehetetlenségi nyomatéka	3.2284E-6 kg.m ²
b a motor viszkózus surlódási konstansa	3.5077E-6 N.m.s
Kb elektromos erő állandó	0.0274 V/rad/sec
Kt motor nyomaték állandó	0.0274 N.m/Amp
R villamos ellenállás	4 Ohm
L elektromos induktivitás	2.75E-6H

Feltételezzük, hogy a rendszer bemenete a motor armatúráján létrehozott feszültség különbség (V), a kimenet pedig a tengely pozíciója théta (θ). A rotor és a tengely merevnek tekinthető, nincs csavarodás, valamint feltételezzük, hogy a súrlódási nyomaték arányos a tengely szögsebességével.

Rendszeregyenletek:

Általában a DC motor által létrehozott nyomaték arányos az armatúra áramerősségével és a mágneses tér erősségével. Ebben a példában azt feltételezzük, hogy a mágneses mező konstans, és ezért a motor nyomatéka csak az „i” armatúra áramával és K_t tényezővel arányos, amint azt az alábbi egyenlet mutatja. Ezt fegyvervezérelt motornak nevezik. $T = K_t i$

Az elektromotoros erő a tengely szögsebességével és egy K_b tényezővel arányos. $e = \theta \dot{K}_b$

Az SI mértékegységben a motor elektromotoros ereje és nyomatéka egyenlőek, ezért a továbbiakban csak egy K együtthatót használunk. A fenti ábra segítségével, felhasználva Newton 2 és Kirchhoff törvényét a következő egyenletet kapjuk:

$$J\ddot{\theta} + b\dot{\theta} = K_i$$

$$L \frac{di}{dt} + R_i = V - K\theta$$

Átviteli függvény:

A fenti két egyenletet Laplace transzformálva:

$$S(Js + b)\theta(s) = KI(s)$$

$$(Ls + R)I(s) = V(s) - Ks\theta(s)$$

A nyílt hurkú átviteli függvényt az I s eliminálásával kapjuk meg úgy, hogy a V armatúra feszültség legyen a bemenet, a kimenet pedig a szögsebesség.

$$P(s) = \frac{\dot{\theta}(s)}{V(s)} = \frac{K}{(Js + b)(Ls + R) + K^2} \left[\frac{\text{rad}}{\frac{\text{sec}}{V}} \right]$$

A feladatban a pozíciót tekintjük kimenetnek, ezért a fenti egyenletet a sebesség integrálásával kaphatjuk meg.

$$\frac{\theta(s)}{V(s)} = \frac{K}{s((Js + b)(Ls + R) + K^2)} \left[\frac{\text{rad}}{V} \right]$$

Állapottér

A fentiekől eltérő differenciálegyenleteket az állapottér alakban is kifejezhetjük úgy, hogy a motor helyzetét, a motor fordulatszámát és az armatúra-áramot választjuk az állapot változóként. Ismét a tápfeszültséget úgy kezeljük, mint bemenetet, és a forgás pozíciót kimenetként választjuk ki.

$$\frac{d}{dt} [\theta \ \dot{\theta} \ i] = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{b}{J} & \frac{K}{J} & 0 & -\frac{K}{L} \\ 0 & -\frac{K}{L} & -\frac{R}{L} & 0 \end{bmatrix} [\theta \ \dot{\theta} \ i] + \begin{bmatrix} 0 & 0 & \frac{1}{L} \end{bmatrix} V$$
$$y = [1 \ 0 \ 0] [\theta \ \dot{\theta} \ i]$$

Követelmények:

Nagyon pontosan szeretnénk elhelyezni a motort, így a motor pozíciójának állandó állapota nulla, ha parancsot adunk. Azt is szeretnénk, hogy az egyensúlyi zavar miatt fennálló egyensúlyi hiba 0 legyen.

A másik teljesítménykövetelmény az, hogy a motor nagyon gyorsan érje el a végső helyzetét túlzott túllendülés nélkül. Ebben az esetben azt szeretnénk, hogy a rendszer 40 ms alatt álljon be, és a túllendülés 16% alatt legyen.

MATLAB PREZENTÁCIÓ:

Az átviteli függvény a MATLAB-ban:

```
J = 3.2284E-6;  
  
b = 3.5077E-6;  
  
K = 0.0274;  
  
R = 4;  
  
L = 2.75E-6;  
  
s = tf('s');  
P_motor = K / (s * ((J*s+b) * (L*s+R) + K^2))  
  
P_motor =
```

0.0274

8.878e-12 s^3 + 1.291e-05 s^2 + 0.0007648 s

Continuous-time transfer function.

Állapot tér:

```
A = [0 1 0  
      0 -b/J K/J  
      0 -K/L -R/L];  
  
B = [0 ; 0 ; 1/L];  
  
C = [1 0 0];  
  
D = [0];  
  
  
motor_ss = ss(A,B,C,D)
```

motor_ss =

A =

	x1	x2	x3
x1	0	1	0
x2	0	-1.087	8487
x3	0	-9964	-1.455e+06

B =

	u1
x1	0
x2	0
x3	3.636e+05

C =

	x1	x2	x3
y1	1	0	0

D =

	u1
y1	0

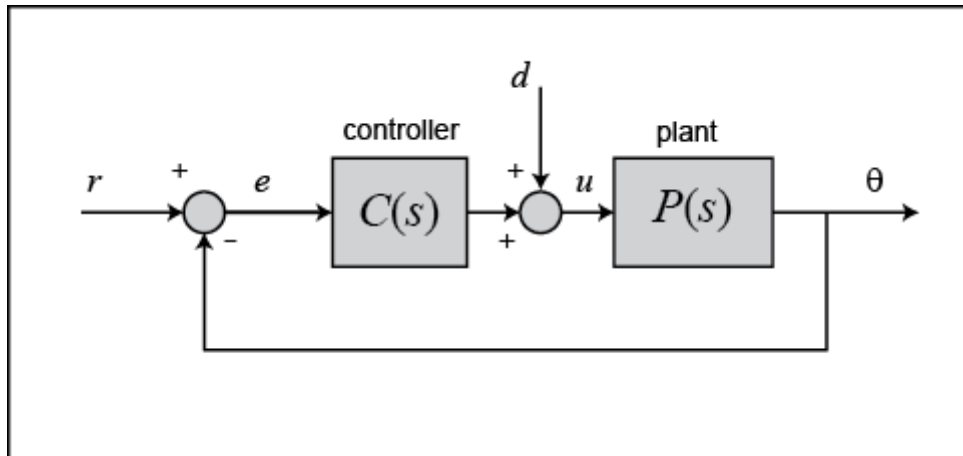
Continuous-time state-space model.

PID szabályozás

A nyílt hurkú átviteli függvényből megkapjuk:

$$P(s) = \frac{\theta(s)}{V(s)} = \frac{K}{s((Js + b)(Ls + R) + K^2)} \left[\frac{\text{rad}}{V} \right]$$

A vezérlési rendszer struktúrája a következő ábrán látható:



2. ábra

A PID szabályzáshoz először létre kell hozni egy m-fájlt MATLAB-ban ahova a következő értékeket visszük be:

```
J = 3.2284E-6;
b = 3.5077E-6;
K = 0.0274;
R = 4;
L = 2.75E-6;

s = tf('s');
P_motor = K / (s * ((J*s+b) * (L*s+R) + K^2));
```

A PID vezérlés átviteli függvénye:

$$C(s) = K_P + \frac{K_I}{s} + K_d s = \frac{K_D s^2 + K_P s + K_I}{s}$$

Az első lépésként a P proporcionális szabályozás értékét kell megfelelően beállítani, hogy eleget tegyen az előírt feltételek mindegyikének.

```
Kp = 1;

for i = 1:3
    C(:, :, i) = pid(Kp);
    Kp = Kp + 10;
end
```

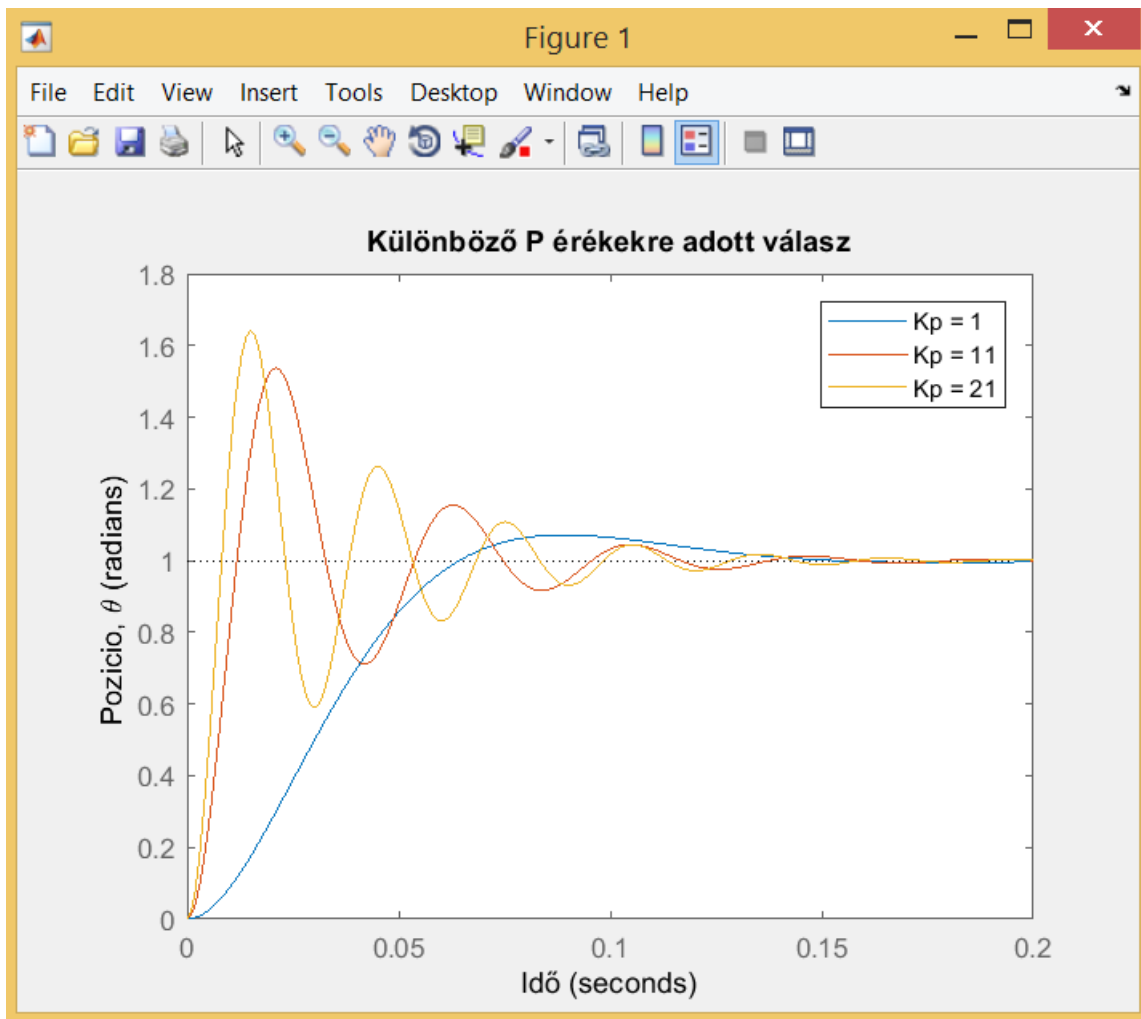


```
sys_cl = feedback(C*P_motor,1);
```

A P értékét 1 és 21 között vizsgáljuk:

```
t = 0:0.001:0.2;  
  
step(sys_cl(:,:,1), sys_cl(:,:,2), sys_cl(:,:,3), t)  
  
ylabel('Pozíció, \theta (radians)')  
xlabel('Idő')  
title('Különböző P értékekre adott válasz')  
legend('Kp = 1', 'Kp = 11', 'Kp = 21')
```

A következő ábrát kapjuk:



3. ábra

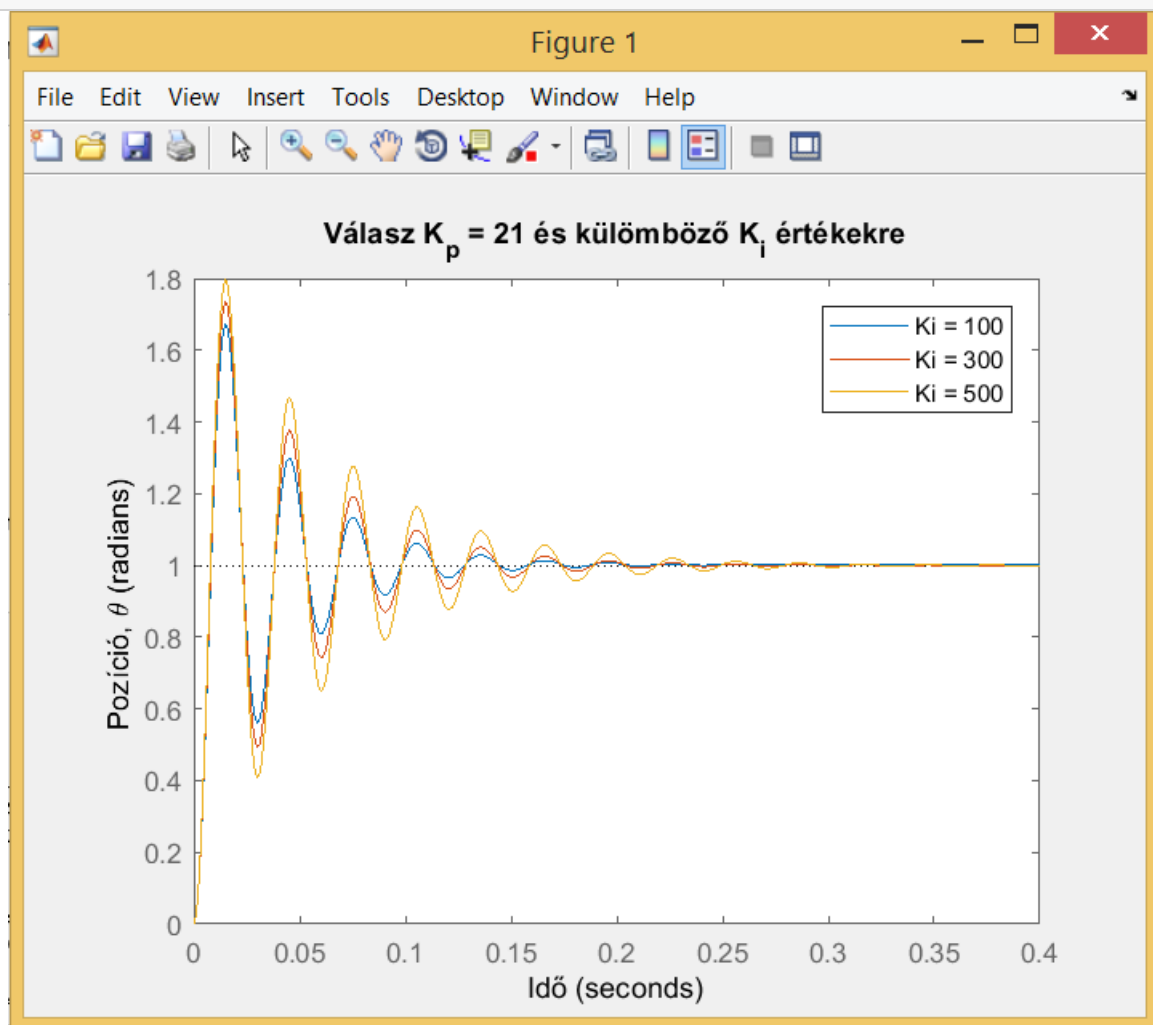
Az ábrán az adott értékek közül számunkra a $K_p = 21$ adott legmegfelelőbb választ, ezért most az I tagot K_i -t kell meghatározni.

```
Kp = 21;

Ki = 100;

for i = 1:5
    C(:, :, i) = pid(Kp, Ki);
    Ki = Ki + 200;
end

sys_cl = feedback(C*P_motor, 1);
t = 0:0.001:0.4;
step(sys_cl(:, :, 1), sys_cl(:, :, 2), sys_cl(:, :, 3), t)
ylabel('Pozíció, \theta (radians)')
xlabel('Idő')
title('Válasz Kp = 21 és különböző Ki értékekre')
legend('Ki = 100', 'Ki = 300', 'Ki = 500')
```



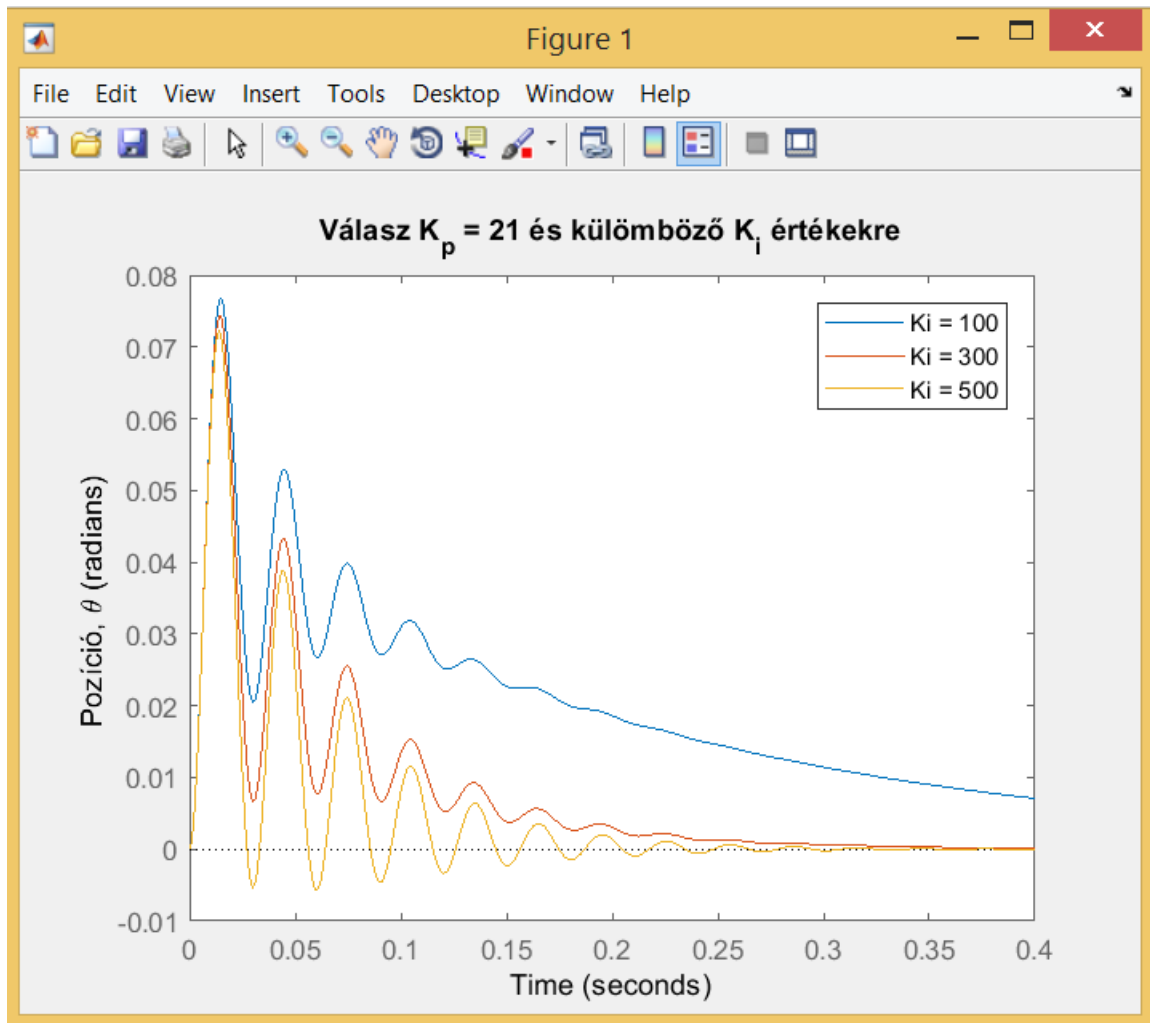
4. ábra

Most meg kell vizsgálni a zavarás értékeinek nagyságát:

```
dist_cl = feedback(P_motor,C);

step(dist_cl(:, :,1), dist_cl(:, :,2), dist_cl(:, :,3), t)

ylabel('Pozíció, \theta (radians)')
title('Válasz Kp = 21 és különböző Ki értékekre')
legend('Ki = 100', 'Ki = 300', 'Ki = 500')
```

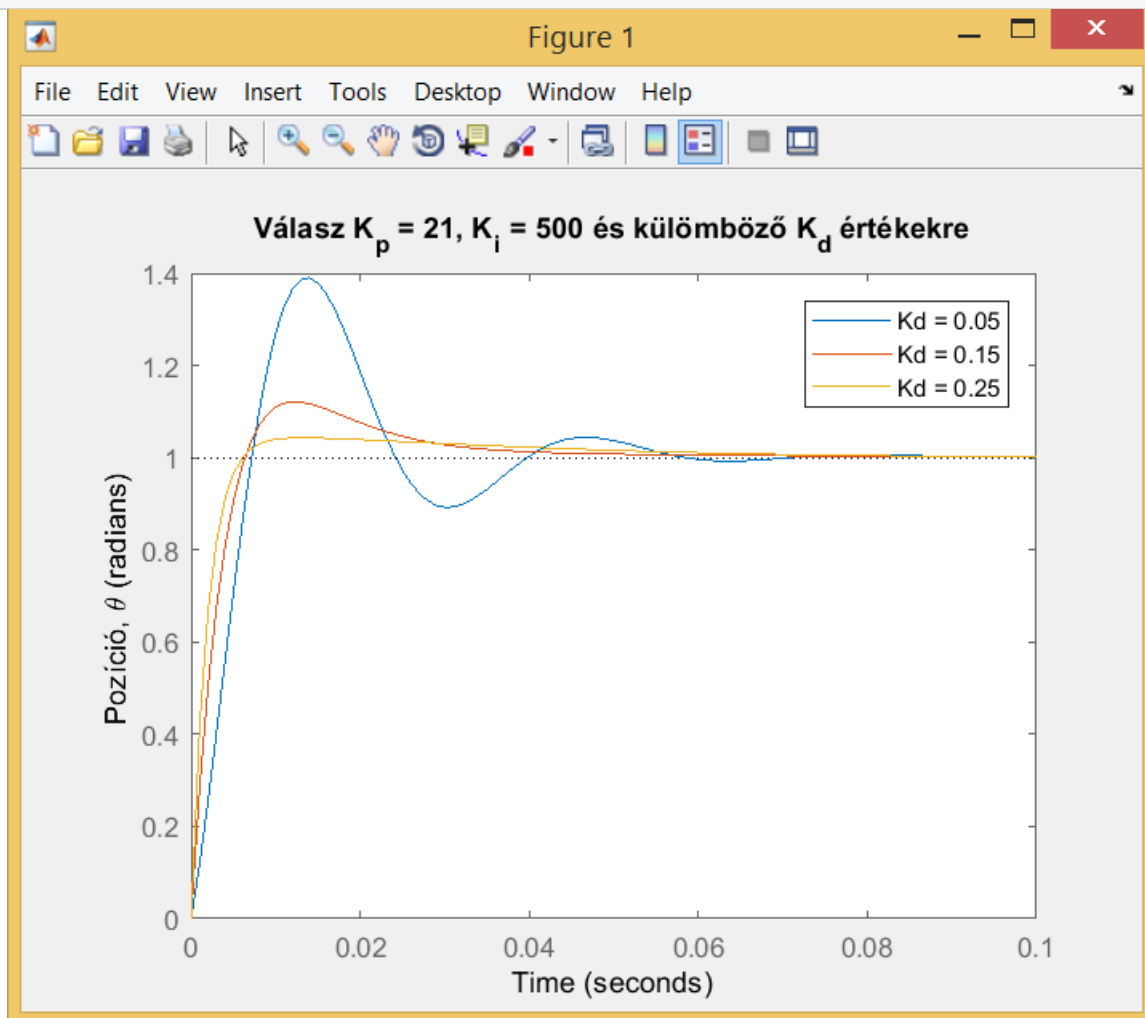


5. ábra

A fenti ábrából látszik, hogy a $K_i=500$ érték felel meg az előírt feltételeknek, mivel ezen érték reagál legjobban a hiba értékére. Az integrált vezérlés csökkentette az egyensúlyi hibát nullára, még akkor is, ha egy lépcsőzetes zavar van; ez volt a cél. Megpróbáljuk csökkenteni az ülepedési időt és a túllövést úgy, hogy derivatív vezérlést adunk hozzá a vezérlőhöz.

A K_d értékét 0,05 és 0,25 között vizsgáljuk.

```
Kp = 21;  
  
Ki = 500;  
  
Kd = 0.05;  
  
for i = 1:3  
    C(:, :, i) = pid(Kp, Ki, Kd);  
    Kd = Kd + 0.1;  
end  
  
sys_cl = feedback(C*P_motor, 1);  
t = 0:0.001:0.1;  
step(sys_cl(:, :, 1), sys_cl(:, :, 2), sys_cl(:, :, 3), t)  
ylabel('Pozíció, \theta (radians)')  
title('Válasz K_p = 21, K_i = 500 és különböző K_d értékekre')  
legend('Kd = 0.05', 'Kd = 0.15', 'Kd = 0.25')
```



6. ábra

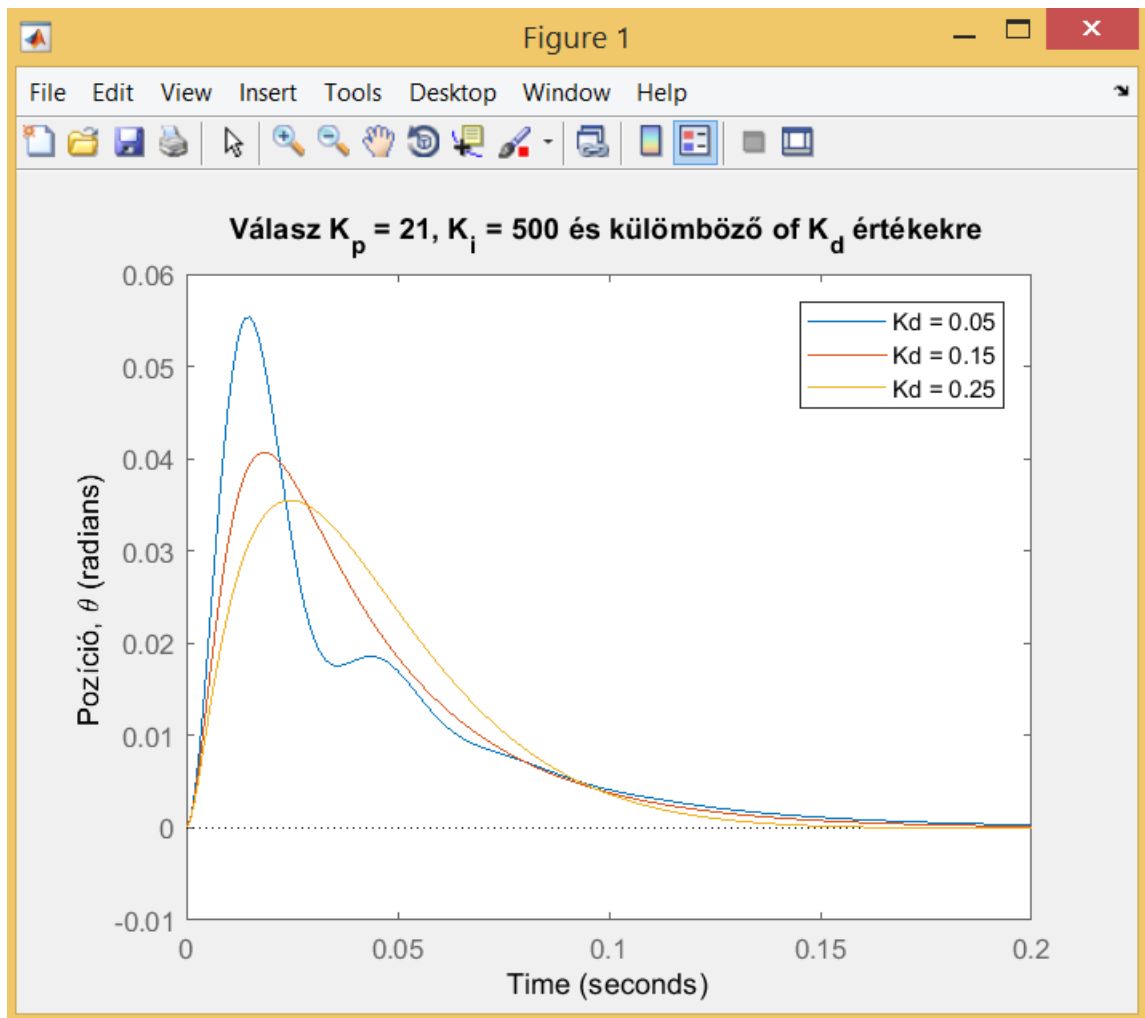
Most figyeljük meg, melyik K_d értékre kapunk megfelelő hiba választ:

```
dist_cl = feedback(P_motor,C);

t = 0:0.001:0.2;

step(dist_cl(:, :,1), dist_cl(:, :,2), dist_cl(:, :,3), t)

ylabel('Pozíció, \theta (radians)')
title('Válasz  $K_p = 21$ ,  $K_i = 500$  és különböző  $K_d$  értékekre')
legend('Kd = 0.05', 'Kd = 0.15', 'Kd = 0.25')
```



7. ábra

Az ábrából leolvasható, hogy a $K_d=0,15$ érték ad elégíti ki legmegfelelőbben az általunk szabott feltételeket. Most nézzük meg a karakterisztikáját stepinfo segítségével:

```
stepinfo(sys_cl(:, :, 2))
```

```
ans =
```

```
    struct with fields:
```

```
    RiseTime: 0.0046
```

```
    SettlingTime: 0.0338
```

```
    SettlingMin: 0.9103
```

```
    SettlingMax: 1.1212
```

```
    Overshoot: 12.1175
```

```
    Undershoot: 0
```

```
    Peak: 1.1212
```

```
    PeakTime: 0.0122
```

Innen látszik, hogy minden feltételünk teljesült, a rendszer 34 ms alatt beáll és a túllendülés 12% os. Tehát a motor pozíció szabályzásának PID értékei: $K_p=21$, $K_i=500$ és $K_d=0,15$.

Adaptív szabályozás:

A mindennapi nyelvhasználatban az "adaptálni" szó jelentése: a viselkedést megváltoztatni, alkalmazkodni az új körülményekhez vagy új állapotokhoz. Ezen értelmezés szerint az a szabályozó tekinthető adaptív szabályozónak, amely képes a viselkedését módosítani, megváltoztatni, válaszolva a folyamat dinamikájának változására, valamint a zavarokra. Mivel a közönséges visszacsatolt rendszer is hasonló megfontolásokból bevezették be, ezért azonnal felmerül az adaptív szabályozás és a visszacsatolt szabályozás közötti különbség kérdése. Évekig sok próbálkozás történt az adaptív szabályozás definiálására. Ezek az erőfeszítések azonban széles körben nem váltak elfogadottá. Hiányzik tehát az adaptív szabályozásoknak egy olyan pontos definíciója, amely lehetővé tenné egy szabályozó hardverének és szoftverének vizsgálatát és eldönteni azt, hogy adaptív-e vagy sem.

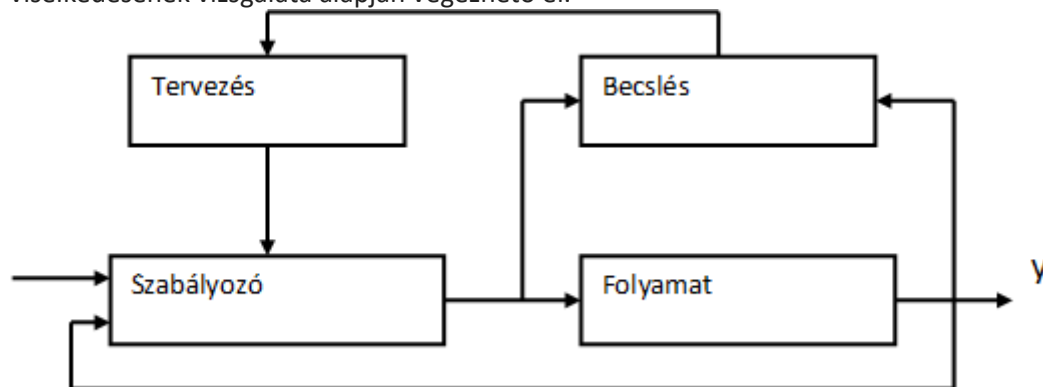
Gyakorlatilag azonban mégis szükséges állást foglalni az adaptív szabályozásokról. Ezért az mondható, hogy az adaptív szabályozás a nemlineáris visszacsatolt szabályozások speciális fajtája, amelyben a folyamatok állapotát olyan két kategóriára lehet szétválasztani, amelyek különböző mértékben változnak. A lassú változási állapotok úgy tekintők, mint paraméterek. Mindez két időskála gondolatával bemutatatható: a gyorsabb időskála a közönséges visszacsatolást jelképezi, a lassabb a szabályozó paramétereinek megváltoztatását mutatja.

Néhány szóban az adaptív szabályozás történetéről és létrejöttéről: az 1950-es évek elején az adaptív szabályozás széles körű kutatása folyt nagyteljesítményű repülőgépek robotpilótáinak tervezésével kapcsolatban. Az ilyen repülőgépek széles sebesség- és magasságtartományban üzemelnek. Úgy találták, hogy a konstans körerősítéses, lineáris, visszacsatolt szabályozók képesek voltak egy

működési állapotban jól működni, de ahogy ezek a működési viszonyok megváltoztak, úgy már nem voltak képesek az előírt jellemzőket tartani. Ezért szükség volt egy olyan igényesebb szabályozóra, amely képes a változó működési viszonyok széles tartományában megfelelően üzemelni. A tárgyban azonban az érdeklődés egyre inkább csökkent, köszönhetően a kutatásokba való bepillantás hiányának, valamint a repülési teszteknel történt szerencsétlenségeknek.

Általában a tervezési cél az, hogy olyan szabályozót válasszunk, amely lehetővé teszi, hogy a szabályozott szakasz - azaz a folyamat - kimenete kövesse a bemeneti referenciajel által előírt jelet. Az általános bemenőjelen és beavatkozó jelen kívül vannak speciális bemenőjelek is, a zavarok. Ezeket nem mi szabályozzuk, mégis befolyásolják a kimenet viselkedését. Ezért a szabályozót úgy kell megtervezni - az előzőek figyelembevételével -, hogy a zavaró hatások a szabályozott szakasz kimenetén már kompenzálódjanak. Tehát a két fő tervezési cél: a pályakövetés és a zavarkompenzáció megvalósítása. Ez minden szabályozási módszernek a központi kérdése.

A szabályozásnak eleget kell tennie a szabályozni kívánt műszaki folyamattal szemben támasztott minőségi követelményeknek. A minőségi követelmények egyrészt a szabályozás statikus pontosságát, azaz állandósult hibáját, másrészt dinamikus viselkedését, azaz szabályozási idejét, túllendülését írják elő. A valóságos és az előírt minőségi jellemzők összehasonlítása a szabályozási rendszer dinamikus viselkedésének vizsgálata alapján végezhető el.



A minőségi követelményeket egy modell segítségével is megfogalmazható, vagyis egy modell segítségével adható meg, hogy különböző alapjelre milyen válasz kívánatos a visszacsatolt rendszertől. Ha a szabályozott szakasz paraméterei állandóak és azokat pontosan ismerjük, akkor pontosan meghatározható, hogy milyen kompenzáló tagra van szükség a megkívánt viselkedés eléréséhez (feltéve, hogy nem támasztunk megvalósíthatatlan követelményeket a rendszerrel szemben). A gondolatsort folytatva eljutunk a modellreferenciás adaptív szabályozáshoz (a későbbiekben részletesen tárgyaljuk). A modellreferenciás szabályozóknál létezik egy szokásos visszacsatolt kört egy kompenzáló taggal. A visszacsatolt szakasz viselkedése egy referenciamodell segítségével definiálható. A modell és a visszacsatolt szakasz ugyanazt az alapjelet kapja meg.

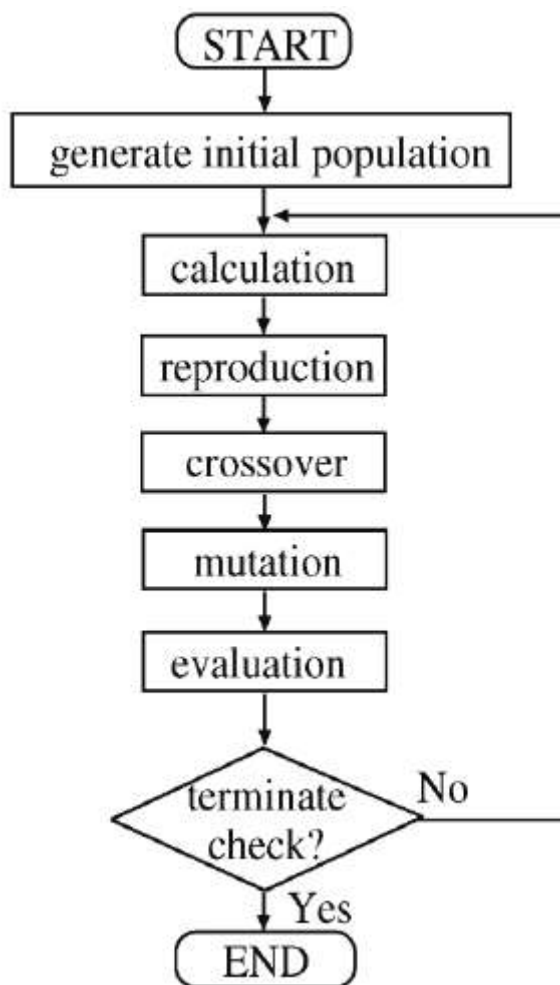
Genetikus algoritmus

A genetikus algoritmus (a továbbiakban: "GA" is jelöli ezt a fogalmat) egy globális optimalizációs eljárás. Bizonyos mértékig kapcsolatban áll az evolúció darwini elméletével. E módszernél egy populáció egyedei (ezek jelképezik a különféle megoldásokat) küzdenek a *túlélésért*. Azonban a túlélést egy egyed számára nem a „gyorsasága”, „ereje”, „leleményessége” (lásd mint a természetben), hanem a feladat megoldásában való eredményessége biztosítja.

A GA hátrányai

Több mint 30 év telt el, amióta John Holland cikkében bevezette a „genetikus algoritmusok” terminust, de úgy látszik egyelőre azóta sem sikerült kiforrott tudományággá válnia az evolúciós számítások kutatásának. Matematikailag még mindig megfoghatatlan miért működik az egyik fitnessfüggvény jól egy adott problémára, míg egy másik ésszerű teljesen rossz eredményt ad.

A genetikus algoritmusok alkalmazásához nagy gyakorlat szükséges, a hatékonyan működő eljárás paramétereinek beállításához sok kísérletezésre van szükség.



A GA előnyei

- Globális optimumot talál
- Diszkrét és folytonos problémáknál egyaránt használhatók
- A különböző adatrepresentációk segítségével a problémák széles területére alkalmazhatók
- Kevés információra van szükségük a problémáról, csak a célfüggvény értékekkel dolgoznak

- Egyszerű algoritmusuk miatt könnyen alkalmazhatók minden problémára és könnyen hibridizálhatók
- Az eljárások párhuzamos számításokra bonthatók, könnyen párhuzamosíthatók
- Egyes problémákra nem ismerünk más megoldást

Hol alkalmazható?

Mindenhol alkalmazható, ahol a feladat sok lehetséges megoldás közül a legjobbat megkeresni. A genetikus algoritmus lehetséges megoldások egy populációját tartja fenn, azaz egyszerre több megoldáskezdeménnyel dolgozik.

Hol alkalmazták már? (Alkalmazási példák)

- Mezőgazdasági termelés optimalizálásában (Debreceni Egyetem Agrártudományi Centrum Agrárgazdasági és Vidékfejlesztési Kar Gazdasági és Agrárinformatikai Tanszék tanulmánya)
- Fák térbeli szerkezetének pontfelhők alapján történő kiértékelésére (Óbudai Egyetem, Alba Regia Műszaki Kar, Geoinformatikai Intézet)

Megvalósítás:

Megvalósítást a google colab programján keresztül Octave-ban és Python nyelvben lett írva.

```
#paraméterek és szükséges programok feltelepítése
```

```
#!pip install slycot # optional
#%matplotlib inline
!pip install control &>/dev/null
!apt install octave &>/dev/null
!pip install oct2py &>/dev/null
!apt install octave-control &>/dev/null
```

```
import control
import numpy as np
from oct2py import octave, Oct2Py
import matplotlib.pyplot as plt
from IPython.display import Image
import math

#plt.style.use('seaborn-whitegrid')
oc=Oct2Py()
oc.eval('pkg load control')
gen=10
```

```
# Genetikus Algoritmus
#értékelés
#fitnessz érték
#keresztelés
egyedek_szama=6
oc.eval('cur=(randi(5, '+str(egyedek_szama)+' ,24))>=3');#8bit mindegyik
    értékre Kp;Ki;Kd 500 egyed
cur=oc.pull('cur')
def lineofarray(k,i):
    oc.push('tmp',k)
    oc.push('j',i)
    oc.eval("tmp2=tmp(j+1,:);")
    return oc.pull("tmp2")
def _calc(_l):
    oc.push('_l',_l)
    oc.eval('tmp=0;for i=1:8 tmp=tmp+(2^(7-(i-1)))*_l(i); end')
    return (oc.pull('tmp'))

def kpid(l):
    _osztó=3
    oc.push('l',l)
    oc.eval('tmp=l(1:8);')
    _Kp=(_calc(oc.pull('tmp'))/_osztó)
    oc.eval('tmp=l(8:16);')
    Ki=( _calc(oc.pull('tmp'))/ osztó)
```

```

oc.eval('tmp=1(16:24);')
_Kd=(_calc(oc.pull('tmp'))/_oszo)
return [_Kp,_Ki,_Kd]
def fitness(t,y): #fitnessz függvény x,y
    tmp=0
    kívánt_érték=1
    felfutási_sebbség=0.03
    túllövés=0.05
    #display(y)
    for i in range(1,len(t)):
        if y[i]<kívánt_érték and t[i]<felfutási_sebbség:
            tmp=tmp+1
        else:
            tmp=tmp-1
            if y[i]+((y[i])*túllövés)>kívánt_érték or y[i]+((y[i])*túllövés)<k
ivánt_érték and t[i]>felfutási_sebbség:
                tmp=tmp+1
            else:
                tmp=tmp-1
            if (y[i]-
(y[i])*túllövés)>kívánt_érték and t[i]>felfutási_sebbség:
                tmp=tmp+1

    return tmp

#kpid(lineofarray(cur,1))
def recomb():
#Recombination
oc.eval("recombined=[];")
oc.eval(str("for i=1:2:length(selectedInds) "
    "r=randi(8); "
    "r1=randi(8); "
    "r2=randi(8); "
    "a=selected(i,:); "
    "b=selected(i+1,:); "

    "c=0; "
    "a1_part=a(1+c:r+c); "#gyerek_part1
    "b1_part=b(r+1+c:8+c); "#gyerek_part2
    "a2_part=b(1+c:r+c); "#gyerek_part1
    "b2_part=a(r+1+c:8+c); "#gyerek_part2
    "#disp(length(a1_part)+length(b1_part)); "

    "#break; "

    "c=8;"
    "c1_part=a(1+c:r+c); "#gyerek_part3
    "d1_part=b(r+1+c:8+c); "#gyerek_part4
    "c2_part=b(1+c:r+c); "#gyerek_part3

```

```

        "d2_part=a(r+1+c:8+c); "#gyerek_part4
        #"disp(length(c2_part)+length(d2_part)); "
        #" end"))
        "c=16;"
        "e1_part=a(1+c:r+c); "#gyerek_part5
        "f1_part=b(r+1+c:8+c); "#gyerek_part6
        "e2_part=b(1+c:r+c); "#gyerek_part5
        "f2_part=a(r+1+c:8+c); "#gyerek_part6
        #"disp(length(e2_part)+length(f2_part)); "
        #" end"))
        "recombined(i,:)=[a1_part,b1_part,c1_part,d1_part,e1_part,f1_part]; "
        " recombined(i+1,:)=[a2_part,b2_part,c2_part,d2_part,e2_part,f2_part]; "
        #"tmp=[a2_part,b2_part,c2_part,d2_part,e2_part,f2_part]; "
        #"disp(length(tmp)); "
        " end"))
def mutation():
    #Mutation
    oc.eval("for i=1:length(selectedInds) "
            "for j=1:8 "
            "if(randi(10)==1) "# 1 a 10 hez mutál egyet
                "recombined(i,j)=xor(recombined(i,j)==1,1)*1; " #xorra
1 bit hely csere
                " end "
            " end "
        " end ")
def selection():
    oc.eval(str("for i=1:length(fitness) "
                "for j=i+1:length(fitness) "
                "if(fitness(i)>fitness(j)) "
                "var=fitness(i); "
                "fitness(i)=fitness(j); "
                "fitness(j)=var; "
                "var=cur(i,:); "
                "cur(i,:)=cur(j,:); "
                "cur(j,:)=var; "
                "end "
                "end "
                "end ")
        "end ")

    oc.eval(str("avgFitness=mean(fitness); "
                "normFitness=fitness/avgFitness; "
                "percFitness=normFitness/length(fitness); "
                "fKum=100*cumsum(percFitness); "
                "rndNums=randi(100,1,length(fitness)) ;"
                "selectedInds=zeros(1,length(fitness)); "))

    oc.eval(str("for i=1:length(fitness) "

```

```

"rnd=rand()*100; "
"for j=1:length(fitness) "
"if fKum(j)>rnd "
" selectedInds(i)=j; "
" break; "
"end "
"end "
"end "));

oc.eval(str("selected=[]; "
"for i=1:length(selectedInds) "
"selected(i,:)=cur(selectedInds(i),:); "
"end "));

```

cur =

```

1 1 1 1 0 1 0 0 1 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0
0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 1 1
0 0 1 1 1 1 0 1 1 1 1 0 1 0 0 1 1 1 0 1 1 1 0 0
1 1 1 0 1 0 0 0 1 1 0 1 1 0 1 1 0 0 1 1 1 0 1 0
0 1 1 1 1 0 0 1 0 0 1 1 0 1 0 1 0 1 1 1 1 1 1 0
0 0 1 0 1 1 0 0 1 1 1 0 0 1 1 1 0 1 1 1 1 0 1 0

```

```

J = 3.2284E-6;
b = 3.5077E-6;
K = 0.0274;
R = 4;
L = 2.75E-6;
oc.push('J',J)
oc.push('b',b)
oc.push('K',K)
oc.push('R',R)
oc.push('L',L)
oc.eval("s=tf('s')")
oc.eval('P_motor = K/(s*((J*s+b)*(L*s+R)+K^2))')

```

Transfer function 's' from input 'u1' to output ...

y1: s

Continuous-time model.

Transfer function 'P_motor' from input 'u1' to output ...

0.0274
y1: -----
8.878e-12 s^3 + 1.291e-05 s^2 + 0.0007648 s

Continuous-time model.

```
def pid_cal(Kp,Ki=0,Kd=0):
```

```
oc.push('Kp',Kp);
```

```

oc.push('Ki',Kd);
oc.push('Kd',Kd);
oc.eval('C = pid(Kp,Ki)')
oc.eval('sys_cl = feedback(C*P_motor,1)')
oc.eval('[y,_t,x]=step(sys_cl,t);')
return([oc.pull('y'),oc.pull('_t')])

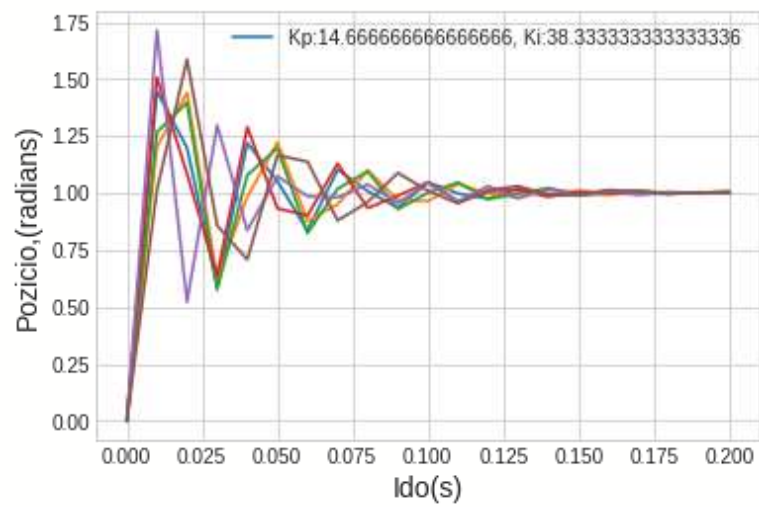
```

```

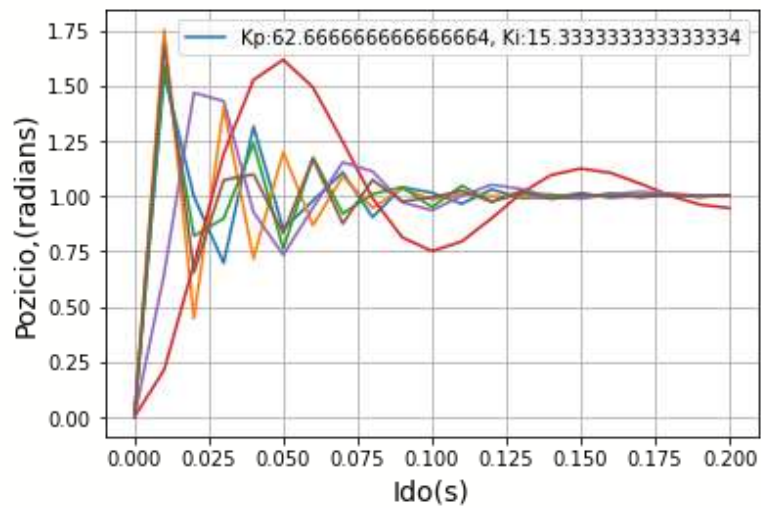
oc.eval("t=0:0.01:0.2;")
t=oc.pull('t')
fig=plt.figure()
#plt.subplot(gen,1,1)
%matplotlib inline
for i in range(1,gen):
    #fig = plt.figure(i)
    fig=plt.figure(i)
    fitness_array=[]
    for j in range(0,len(cur)):
        [Kp,Ki,Kd]=kpid(lineofarray(cur,j))
        [y,_t]=pid_cal(Kp,Ki,Kd)
        #plt.subplot(gen,1,i)
        plt.plot(_t,y) #x,y
        fitness_array.append(fitness(_t,y))
        #display('Fitnessz '+str(fitness_array[j-1]))
        plt.legend(["Kp:"+str(Kp)+", Ki:"+str(Ki)] #+", Kd:"+str(Kd)])
    oc.push('fitness',fitness_array)
    oc.eval("[u,v]=max(fitness);")
    oc.eval("elit=cur(v,:);")
    oc.eval("disp(cur)")
    selection()
    recomb()
    mutation()
    oc.eval("cur=recombined")
    #oc.eval("disp(cur)")
    #oc.eval("disp(elit)")
    fig.suptitle('Különböző PI értékekre adott válasz', fontsize=20)
    plt.xlabel(str("Ido(s)"),fontsize=14)
    plt.ylabel(str("Pozicio,(radians)"),fontsize=14)
    plt.grid('both')
    plt.yscale("linear")
    fig=plt.figure()
    fig.savefig("plot"+str(i)+".png")
#break

```

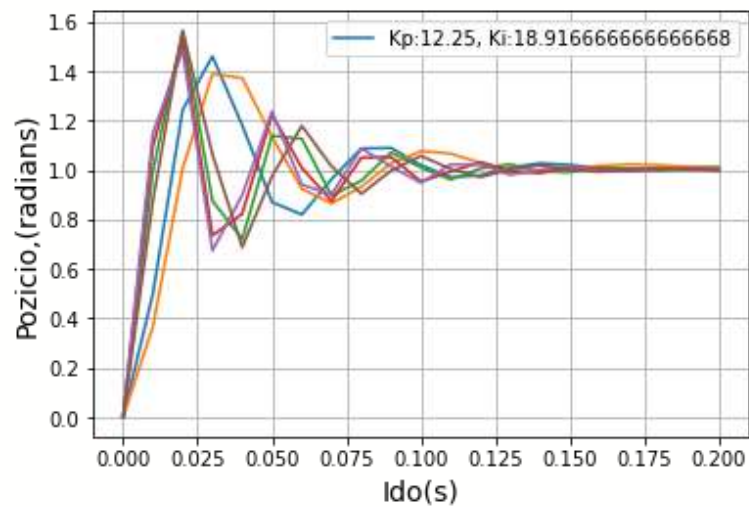
Különböző PI értékekre adott válasz

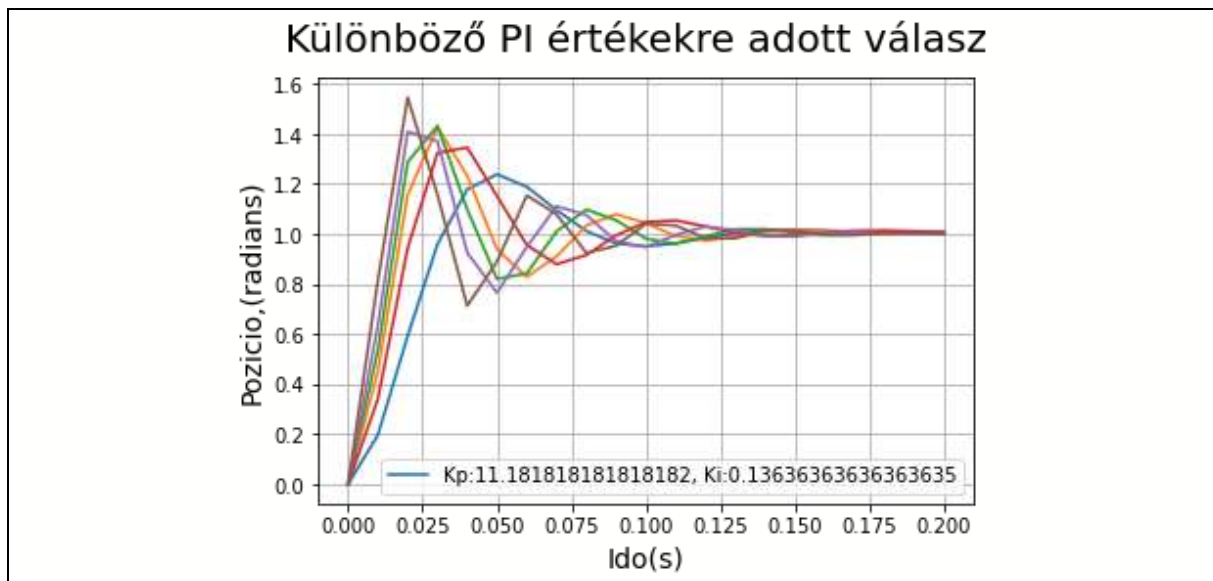


Különböző PI értékekre adott válasz



Különböző PI értékekre adott válasz





Összegzés

A program 10 generáción keresztül futott, és eredményt is kaptunk. Az, hogy ez optimális vagy nem, az más kérdés. Viszont ezt határoztuk meg kilépési feltételnek.

A szemiáriumi munkáról talán kijelenthető az, hogy *sikerként* tudjuk elkönyvelni.

Felhasznált irodalom:

<http://navneetvk.blogspot.com/2009/08/self-tuning-pid-algorithm.html>