



Mechatronikai rendszerek modellezése bakteriális evolúciós algoritmussal

HABA TAMÁS

Mechatronikai mérnöki MSc. hallgató

Témavezetők:

DR. BUDAI CSABA

adjunktus

Mechatronika, Optika és Gépészeti Informatika Tanszék (BME-GPK)

DR. BOTZHEIM JÁNOS

egyetemi docens

Mesterséges Intelligencia Tanszék (ELTE-IK)

TDK dolgozat

2021

Tartalomjegyzék

1. Bevezetés	5
2. Bakteriális evolúció	9
2.1. Evolúciós algoritmusok	9
2.2. A bakteriális evolúció alapjai	9
2.3. Szimbolikus regresszió	11
2.3.1. Egyedek reprezentációja	11
2.3.2. Egyedeken alkalmazható műveletek	12
2.4. Implementáció	15
2.4.1. Az elkészült program használata	15
2.4.2. Optimalizáció	19
3. Mechatronikai alkalmazás	21
3.1. Autonóm dinamikai rendszerek modellezése	21
3.2. Rendszeridentifikáció	24
3.2.1. A vizsgált rendszer	25
3.2.2. Klasszikus identifikációs eljárások	25
3.2.3. Indentifikáció bakteriális evolúcióval	27
3.3. Modellprediktív szabályozás	29
3.3.1. A szabályozott szakasz modellje	30
3.3.2. Prediktor tervezése klasszikus megközelítés alapján	31
3.3.3. Prediktor tervezése bakteriális evolúciós algoritmussal	33
3.3.4. A prediktív szabályozás szimulációja	35
4. Összefoglalás	39
5. Felhasznált irodalom	41

1. Bevezetés

A mechatronika egyik fő célja összetett rendszerek és folyamatok irányítása, mely során különböző típusú, például mechanikai, villamos és hidraulikus alrendszerek kapcsolódnak össze, gyakran a mérési adatgyűjtés, a jelfeldolgozás és a számítógépes irányítás eszközeivel együttesen. Ilyen rendszerek tervezése során az elsődleges feladat az irányítani kívánt folyamat modellezése és az azt jellemző dinamikai viselkedés megfelelő pontosságú leírása.

A matematikai modellezés során legtöbbször közönséges differenciálegyenletek formájában kerül leírásra a rendszer viselkedése. Ez azonban nem tehető meg minden esetben egyszerűen: a mérnöki gyakorlatban sokszor szükséges nemlineáris jelenségeket figyelembe venni, mint például a száraz súrlódás hatásait [1], vagy a fizikai megvalósításból eredő egyéb nem ideális viselkedést, például a beavatkozó szervek telítődését, vagy a számábrázolás okozta térbeli kerekítését, az úgynevezett kvantálást. Emellett gyakran megjelenik kapcsolt dinamikai jelleg is mintavételes rendszerek esetén, hiszen a digitális eszközök csak adott órajellel szinkronizálva képesek a folytonos idejű folyamatokat megfigyelni és azokba beavatkozni [2]. Még ha fel is írható a rendszer dinamikáját leíró differenciálegyenlet, annak megoldására előfordulhat, hogy nem létezik zárt alakú, analitikus megoldás.

A modellezés nehézségei különböző módszerekkel kerülhetők el. A nemlineáris hatások egyensúlyi helyzet körüli vagy munkaponti linearizálás segítségével vehetők figyelembe [3]. Sokszor egy adott jelenség valamilyen közelítéssel élve könnyebben modellezhető, mint például a digitális szabályozásban a mintavételezés időkéséssel való közelítése [4–6]. Bizonyos esetekben a matematikai nehézségeket okozó hatások teljesen el vannak hanyagolva. Közelítő módszerek nem csak a modell felállításakor, hanem annak megoldásakor, kiértékelésekor is alkalmazhatók, például nemlineáris differenciálegyenletek megoldására számos széles körben elterjedt numerikus módszer [7] került kifejlesztésre.

Fontos látni, hogy a modell pontossága és a szükséges erőforrások között kompromisszumot kell kötni. A túl sok elhanyagolás és közelítés azt eredményezi, hogy a vizsgált jelenségeket a modell pontatlanul írja le, azonban az összetettebb modellek numerikus megoldása nagyobb számítási teljesítményt igényel. Ez a probléma új megközelítéssel, gépi tanulási módszerek alkalmazásával megoldhatóvá válhat.

A különböző gépi tanulási módszerek alapjai már a 20. században megszülettek, ide sorolhatók többek között az evolúciós algoritmusok [8], a fuzzy rendszerek [9] és a neurális hálózatok [10, 11] is. Azonban a 21. századra vált elérhetővé olyan mértékű számítási teljesítmény, amellyel ezen módszerek széles körben hasznosíthatók lettek [12]. A gépi tanulási módszerekkel olyan problémák váltak megoldhatóvá, amelyeket klasszikus megközelítésekkel addig nem, vagy csak nagyon nehezen lehetett megoldani. Az elmúlt években számos ehhez köthető átütő erejű eredmény született, például a mintafelismerés terén [13] vagy a Google AlphaGo [14] programja, mely elsőként győzött le profi emberi Go játékost.

Számos műszaki területen, így a mechatronikában is megtalálható a gépi tanulás alkalmazása. Ez leggyakrabban a szabályozás megvalósításában nyilvánul meg, például neurális hálózaton alapuló [15] vagy fuzzy szabályhalmazon alapuló szabályozás [16] során. Bizonyos módszerek azonban háttérbe szorultak, mint például a bakteriális evolúciós algoritmusok.

A bakteriális evolúciós algoritmusok a genetikai algoritmusokkal szemben nem az állatvilág, hanem a baktériumok törzsfajlásához hasonló módon oldanak meg optimalizációs problémákat [17]. Ennek alapja a bakteriális mutáció és a kizárólag baktériumokra jellemző, egyedek közötti horizontális géntranszfer [18]. Bizonyos problémák esetén, ahol egyaránt alkalmazható genetikai és bakteriális evolúciós algoritmus, utóbbiak hatékonyabbnak bizonyultak [19]. Ezen felül bakteriális evolúcióval úgynevezett szimbolikus regressziós problémák [20] is megoldhatók, ahol a megoldásteret matematikai kifejezések alkotják, melyek struktúrája és bonyolultsága előre nem ismert [21]. Ez nagy előny a mechatronikai rendszerek modellezése során, hiszen gyakran nem áll rendelkezésre olyan konstruktív matematikai módszer, amivel a vizsgált rendszer modellje előállítható lenne.

Ezen dolgozat fő célkitűzése a bakteriális evolúciós algoritmusok alkalmazási lehetőségeinek vizsgálata a mechatronikában. A dolgozatban bemutatásra kerül egy bakteriális evolúciós módszer, amellyel dinamikai rendszerek modellezhetők, valamint egy saját fejlesztésű implementáció, amellyel akár GPU-val gyorsított tanítás végezhető számos konfigurálható paraméterrel. A módszer alkalmazási lehetőségei mérési eredményeken alapuló modell identifikáció, valamint modell prediktív szabályozás példáján keresztül kerülnek ismertetésre.

A dolgozat három fő fejezetből áll. A 2. fejezet bemutatja a bakteriális evolúciós algoritmust és a saját implementáció részleteit. A 3. fejezet kísérleti és szimulációs eszközökön keresztül mutatja be a mechatronikai alkalmazási lehetőségeket, végül a 4. fejezet összefoglalja az elért eredményeket.

2. Bakteriális evolúció

Ez a fejezet a bakteriális evolúciós algoritmusok általános érvényű alapvetéseit, valamint a módszer alkalmazásának speciális esetét tárgyalja szimbolikus regressziós problémákon. Ezen felül bemutatásra kerül egy saját készítésű implementáció is.

2.1. Evolúciós algoritmusok

Az evolúciós algoritmusok olyan optimalizációs módszerek, melyek egy adott problémához a biológiai evolúcióhoz hasonló módon állítják elő a megoldást, folyamatos fejlődés révén. Az evolúciós algoritmusok legkisebb önálló egységei az *egyedek* [22], melyek egy-egy lehetséges megoldást jelentenek az optimalizálandó problémára. Az egyedek sokasága alkotja a *populációt* [22], amely az evolúció során generációról generációra fejlődik.

Ezt a fejlődést különféle eszközökkel lehet megvalósítani, ami alapján az evolúciós algoritmusokat további kategóriákba lehet sorolni. Ilyenek például a genetikus algoritmusok [22], melyek a darwini evolúcióra épülnek: a populáció egyedei szaporodnak és utódokat hoznak létre, melyek valamilyen módon az ősök tulajdonságait keresztezve viszik tovább illetve véletlenszerűen mutálódhatnak. A folyamatos fejlődést a szelekció biztosítja, amely során a jobb egyedeknek van nagyobb esélyük a túlélésre. A genetikus algoritmusok széles körben elterjedtek, azonban léteznek más evolúciós megközelítések is, mint például a bakteriális evolúció [17].

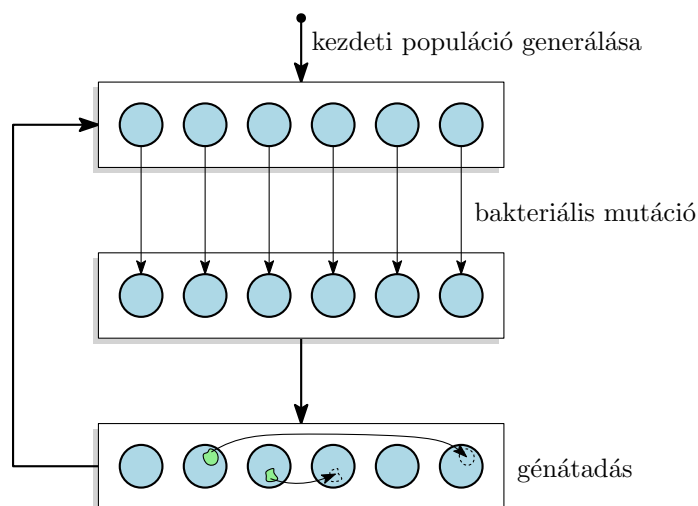
2.2. A bakteriális evolúció alapjai

A bakteriális evolúciós algoritmusokat „biológiai” baktériumok ihlették, azok ugyanis képesek horizontális génátadásra, azaz az örökítőanyag és az abban kódolt információ az állatvilágtól eltérően nem csak a felmenő és az utód között (vertikálisan) történik, hanem adott generáción belül két baktérium között is létrejöhet. Így egy adott egyed

képes akár élete során is átvenni kedvező tulajdonságokat a környezetétől, megfelelve az aktuális életfeltételeknek. Emiatt alkalmazkodó képességük kiváló, többek között ezért tudnak a kórokozó baktériumok hamar rezisztenssé válni egy adott antibiotikumra. Ez az adaptációs képesség hívta életre a bakteriális evolúciós algoritmusokat.

A bakteriális evolúciós algoritmus lépései az alábbiak:

1. **Kezdeti populáció létrehozása:** A bakteriális evolúciós algoritmus inicializálása során az egyedek véletlenszerűen jönnek létre, ezzel megalkotva a kezdeti populációt.
2. **Bakteriális mutáció:** Minden egyes egyedhez létrejön adott számú klón. Az egyes klónok véletlenszerűen kiválasztott részei véletlenszerűen módosulnak. Adott jósági (fitness) vagy hibafüggvény alapján a legjobb klón az eredeti helyébe lép, amennyiben jobb nála.
3. **Génátadás:** A populáció egy adott jósági vagy hibafüggvény alapján két félre tagozódik, jó és rossz egyedekre. Véletlenszerűen kiválasztásra kerül adott számú jó és rossz egyed. A jó egyedek ezután a hozzájuk kiválasztott rossz egyedeknek gént adnak át, azaz egy véletlenszerűen kiválasztott részükkel részlegesen felülírják azokat.
4. **Kilépési feltétel vizsgálata** Amennyiben a legjobb egyed az optimalizációs problémára elfogadható megoldást ad, úgy az evolúció véget ér. Ellenkező esetben a mutációs lépéstől folytatódik.



2.1. ábra. A bakteriális evolúciós algoritmus főbb lépései.

A bakteriális evolúció lépéseit a 2.1 ábra szemlélteti. Ez a leírás önmagában nem elegendő egy konkrét implementáció elkészítéséhez, hiszen több részlet tisztázatlan maradt: az egyedek reprezentációja, véletlenszerű létrehozásuk és módosításuk, kiértékelésük és összehasonlításuk menete stb. A megvalósítás pontos módja nagyban függ az optimalizálandó feladat jellegétől, ami a továbbiakban kerül ismertetésre.

2.3. Szimbolikus regresszió

A szimbolikus regresszió az a folyamat, amelynek célja előállítani valamilyen adatsorra legjobban illeszkedő modellt. Ilyenkor a keresési tér állhat például algebrai kifejezések közül, amelyek összetett, hierarchikus szerkezetűek, egymástól akár teljesen eltérő bonyolultságúak.

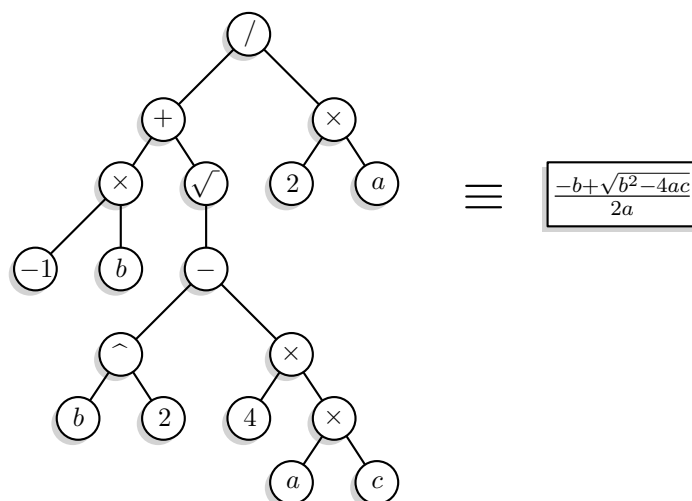
Szimbolikus regressziós problémára vezethető vissza számos olyan mérnöki feladat, amely során nem rendelkezünk előzetes információval a modellezni kívánt rendszer struktúrájáról és bonyolultságáról. A továbbiakban bemutatásra kerül, hogy milyen módon lehet bakteriális evolúciós algoritmussal szimbolikus regressziós problémákat megoldani.

2.3.1. Egyedek reprezentációja

Az egyes egyedek algebrai kifejezések, melyek tartalmazhatnak konstansokat, változókat, különféle műveleteket és függvényeket. A műveleti sorrend meghatároz egy adott hierarchiát, ami alapján egy algebrai kifejezés eredménye további rész-kifejezések értékétől függ.

Fagráfok segítségével ez a hierarchia könnyen reprezentálható, ahogy azt a 2.2. ábra szemlélteti. Ahogy látható, minden csúcshoz egy adott művelet tartozik, ami a belőle kiinduló részfák eredményét dolgozza fel. Ezeket a fákat *kifejezésfáknak* [23] nevezik.

A kifejezésfák levelei a *terminális kifejezések* [23], amik lehetnek konstansok vagy változók. A további műveletek az operandusaik száma alapján csoportosíthatók. Az *unáris* műveleteknek egy operandusuk van, mind például a négyzetgyökvonás vagy a $\sin()$, $\cos()$, $\exp()$ függvények. A *bináris* műveletek kétoperandusúak, ide tartozik a négy alapművelet. Lehetséges *ternáris* műveleteket is definiálni, például feltételes elágazást, ahol a feltétel, az igaz ág és a hamis ág a három operandus.



2.2. ábra. Egy algebrai kifejezés reprezentálása kifejezésfa segítségével.

2.3.2. Egyedeken alkalmazható műveletek

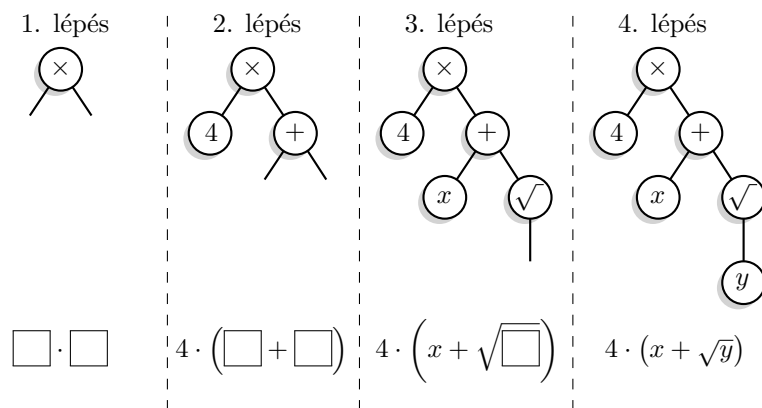
Az egyedek reprezentálása mellett definiálni kell azt is, hogy milyen módon lehet azokat módosítani és hogyan valósíthatók meg a bakteriális evolúció egyes lépései. A kifejezésfák önhasznós tulajdonsággal bírnak, hiszen a részkifejezések önmagukban is tekinthetők egy önálló kifejezésnek. Ennek következtében a kifejezésfák rekurzív algoritmusokkal könnyen kezelhetők.

Generálás

A generálás célja egy véletlenszerű egyed létrehozása, amire először a kezdeti populáció létrehozásánál van szükség. Könnyen generálhatók kifejezésfák rekurzív módon az 1. algoritmus alapján. A fa generálása egy véletlenszerű gyökérelmé létrehozásával kezdődik, amelyből a teljes kifejezésfa kiindul. Amennyiben ez nem terminális kifejezés, úgy a szükséges számú operandust is ugyanúgy kell generálni, mint teljes kifejezésfát. A módszert a 2.3. ábra szemlélteti.

Fontos figyelembe venni azt is, hogy a generálás során akár nagyon terebélyes fák is létrejöhetnek, ami felesleges komplexitást vihet a populációba, valamint ezek feldolgozása megnövelheti a számítási igényt. Ez a probléma orvosolható a fák mélységének és szélességének korlátozásával úgy, hogy bizonyos korlátokat átlépve már csak terminális kifejezések jöhetnek létre.

A generálás könnyen paraméterezhető azzal, ha előre meg van szabva, hogy milyen típusú művelet mekkora valószínűséggel jöhet létre. Ezen felül a mélységre és szélességre vonatkozó korlátok is lehetnek generálási paraméterek.



2.3. ábra. Véletlenszerű kifejezésfa generálásának folyamata és az így kapott algebrai kifejezés.

1. Algoritmus. Véletlenszerű kifejezésfa generálása.

```

function VÉLETLENSZERŰKIFEJEZÉSFA
    gyökérElem ← VéletlenszerűMűvelet()
    for  $i \leftarrow 1$  to gyökérElem.gyermekekSzáma do
        gyökérElem.gyermekek[i] ← VéletlenszerűKifejezésfa()
    end for
    return gyökérElem
end function

```

Bakteriális mutáció

A bakteriális mutáció során egy adott egyedet klónozni kell, azaz másolatokat kell létrehozni belőle. A fák másolása szintén rekurzív módon végezhető a 2. algoritmus alapján, a gyökérelem és a részfák másolásával.

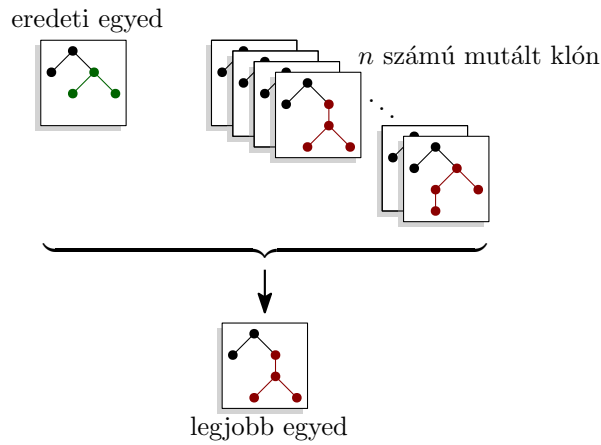
2. Algoritmus. Kifejezésfa másolása.

```

function KIFEJEZÉSFA MÁSOL(eredeti)
    másolat ← eredeti
    for  $i \leftarrow 1$  to eredeti.gyermekekSzáma do
        másolat.gyermekek[i] ← KifejezésfaMásol(eredeti.gyermekek[i])
    end for
    return másolat
end function

```

A klónok mutálásának egy lehetséges módja a fa egy ágának véletlenszerű kiválasztása, és abból kiindulva új részfák növesztése a 3. algoritmus szerint ismertetett módon. A bakteriális mutációt a 2.4. ábra mutatja.



2.4. ábra. Bakteriális mutáció értelmezése kifejezésfákon.

3. Algoritmus. Bakteriális mutáció.

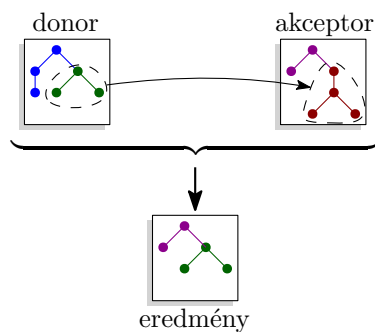
```

function BAKTERIÁLISMUTÁCIÓ(egyed, klónokSzáma)
  k ← VéletlenszerűÁgIndex(egyed)
  klónok[klónokSzáma]
  for i ← 1 to klónokSzáma do
    klónok[i] ← KifejezésfaMásol(egyed)
    klónok[i].ágak[k] ← VéletlenszerűKifejezésfa()
  end for
  egyed ← LegjobbKiválaszt(egyed, klónok)
end function

```

Génátadás

A génátadás két egyed, a donor és az akceptor között jön létre. Ki kell választani az akceptorból egy véletlenszerű részfat (hasonlóan a mutációhoz), ezt azonban törölni kell, és a helyére be kell másolni a donorból véletlenszerűen kiválasztott részfat. A génátadást a 4. algoritmus és a 2.5. ábra szemlélteti.



2.5. ábra. Génátadás értelmezése kifejezésfákon.

4. Algoritmus. Génátadás.

```
function GÉNÁTADÁS(donor, akceptor)
  k ← VéletlenszerűÁgIndex(donor)
  l ← VéletlenszerűÁgIndex(akceptor)
  akceptor.ágak[l] ← KifejezésfaMásol(donor.ágak[k])
end function
```

2.4. Implementáció

Az előzőekben bemutatott bakteriális evolúciós algoritmus implementálásra került C++ nyelven. A forráskód semmilyen platformspecifikus részletet nem tartalmaz, így különféle platformokra lefordítható, a program tesztelések során Windows és Linux környezetben is problémamentesen futott.

2.4.1. Az elkészült program használata

A program egy futtatható állományból áll, amit terminálból indítva lehet használni. Indításkor megkeresi az aktuális könyvtárban a konfigurációs fájlt, betölti a tanítási fájlt, majd elkezd az evolúciós algoritmus futtatását.

Tanítási fájl és kiértékelés

A bakteriális evolúciós algoritmus felügyelt tanulás lévén [17, 22] az optimalizációt előre ismert tanítási minták alapján végzi, melyek bemeneteket és a hozzájuk tartozó elvárt kimenetet tartalmaznak.

A szimbolikus regressziós feladat során az egyedek algebrai kifejezések, amelyeknek az egyes változói tekinthetők bemenetnek, maga a kiértékelt kifejezés pedig a kimenet. A tanítási fájl struktúráját a 2.1. kód mutatja.

t	lambda	x0
0	-5	-1 -1
0	-5	-0.8 -0.8
0	-5	-0.6 -0.6
...		

2.1. kód. A tanítási fájlok felépítése.

Az első sor tartalmazza a fejléct, ahol a lehetséges változónevek szerepelnek üres helyekkel (szóközökkel vagy tabulátorral) elválasztva. Ezután minden sor egy tanítási

mintát tartalmaz, ami a változók helyettesítési értékeit jelenti. A fejlécben nevesített oszlopokon kívül az utolsó, külön nem megnevezett oszlop tartalmazza az elvárt kimeneti értéket.

Egy adott kifejezésfa kiértékelése a tanítási fájl alapján történik. A kiértékelendő egyed megkapja a tanítási mintákban található behelyettesítési értéket, és az elvárt kimenettől való eltérésből meghatározható e hiba értéke

$$e = \sqrt{\frac{1}{n} \sum_{i=0}^n (f(\mathbf{x}_i) - y_i)^2}, \quad (2.1)$$

ahol n a tanítási minták száma, f a kiértékelte kifejezésfa által reprezentált többváltozós függvény, $\mathbf{x}_i$ a helyettesítési értékek vektora és y_i az elvárt kimenet. Ez a hibaszámítás a négyzetes hiba vagy RMSE (root mean square error). Az evolúció során a cél ezen hiba minimalizálása.

Konfigurációs fájl

A bakteriális evolúció különböző paramétereit a program egy JSON formátumú konfigurációs fájlból olvassa ki, ami alapértelmezetten az aktuális munkakönyvtárban található `config.json` fájl. Ennek struktúráját a 2.2. kód mutatja.

Az egyes paraméterek jelentése:

- `N_ind`: a populáció egyedszáma
- `N_inf`: génátadások száma egy generáción belül
- `N_clones`: klónok száma egy egyed bakteriális mutációja során
- `max_iter`: maximális generációk száma, ami után az evolúció leáll
- `min_rmse`: minimális hiba, aminek elérése után az evolúció leáll
- `train_file`: a tanítási fájl elérési útonala
- `population_file`: az egyedeket tartalmazó fájl elérési útvonala
- `save_interval`: egyedek mentési gyakorisága (másodpercben)
- `log_interval`: naplózási gyakoriság a standard kimenetre (másodpercben)
- `evaluators`: az egyedeket kiértékelő szálak (CPU-n és GPU-n egyaránt)
- `max_depth`: kifejezésfa maximális generálási mélysége
- `vars`: kifejezésfában szereplő lehetséges változók nevei
- `const_min`: konstansok lehetséges minimális értéke
- `const_max`: konstansok lehetséges maximális értéke
- `probability`: különböző típusú műveletek generálási valószínűségének súlyai

```

{
    "N_ind": 50,
    "N_inf": 5,
    "N_clones": 3,

    "max_iter": 30000,
    "min_rmse": 1e-5,

    "train_file": "train.txt",
    "population_file": "population.txt",
    "save_interval": 5,
    "log_interval": 0,

    "evaluators": [
        { "type": "CPU" },
        { "type": "GPU",
          "OpenCL_platform": 0,
          "OpenCL_device": 0 },
        ...
    ],

    "max_depth": 5,
    "vars": [ "t", "x0", "lambda" ],
    "const_min": -10,
    "const_max": 10,
    "probability": {
        "const": 6,
        "var": 6,
        "+": 1,
        "-": 1,
        ...
    }
}

```

2.2. kód. A konfigurációs JSON fájl felépítése.

Naplózás

A konfigurációs fájlban megadott naplózási gyakoriság arra vonatkozik, hogy a standard kimeneten (terminálon) legalább hány másodpercenként jelenítse meg a program a futás állapotát. Amennyiben 0 másodperc van megadva, úgy minden generáció során megtörténik ez a kiírás 2.3. kód szerint.

```
Running bacterial evolution:
EPOCH #0 best RMSE=3.88658
EPOCH #1 best RMSE=3.88658
EPOCH #2 best RMSE=3.88656
EPOCH #3 best RMSE=3.87806
EPOCH #4 best RMSE=3.87806
EPOCH #5 best RMSE=3.87806
EPOCH #6 best RMSE=3.87806
...
```

2.3. kód. A program kimenete a futtatás során.

Ennek során az eltelt generációk száma (EPOCH) és a legjobb egyed hibája kerül kiírásra. A naplózás fájlba is könnyen megtehető a standard kimenet eltérítése vagy például a tee UNIX parancs segítségével.

A populáció egyedei bizonyos időközönként mentésre kerülnek a konfigurációs paraméterek szerint megadott fájlba. A kifejezésfák hierarchikus szerkezetét szövegesen *prefix* alakú kifejezéseként lehet tárolni 2.4. kód szerint. Minden csomópontot az adott kifejezés típusa ír le, melyet zárójelben az adott csomópontból kiinduló részfák felsorolása követ szóközzel elválasztva. Az implementált műveletek prefix alakja megtalálható a 2.1. táblázatban.

```
var(i0)
/(var(i0) const(3.17972))
+(var(u1) exp(const(2)))
*(var(y) +(const(-2) var(x)))
...
```

2.4. kód. Egyedek prefix formátumú reprezentációja a populációs fájlban.

2.1. táblázat. Az implementált műveletek és jelöléseik.

	művelet	matematikai jelölés	prefix alak
terminális kifejezés	konstans	3, 14	const(3.14)
	változó	a	var(a)
unáris művelet	exponenciális	$e^{\dots}$	exp(...)
	szinusz	$\sin(\dots)$	sin(...)
	koszinusz	$\cos(\dots)$	cos(...)
	szignum	$\text{sgn}(\dots)$	sgn(...)
bináris művelet	összeadás	$\dots + \dots$	+(... ..)
	kivonás	$\dots - \dots$	-(... ..)
	szorzás	$\dots \cdot \dots$	*(... ..)
	osztás	$\dots / \dots$	/(... ..)
ternáris művelet	feltétel	$\begin{cases} \dots & \text{ha } \dots > 0 \\ \dots & \text{egyébként} \end{cases}$	if(... ..)

2.4.2. Optimalizáció

A tanulás során előfordulhat, hogy a vizsgált rendszer komplexitása (például sok változó) miatt a tanítási minták száma igen magas, akár százezres vagy milliós nagyságrendbe esik. Ezzel együtt az egyedek kiértékelése jól párhuzamosítható feladat, ami lehetőséget nyújt az optimalizálásra.

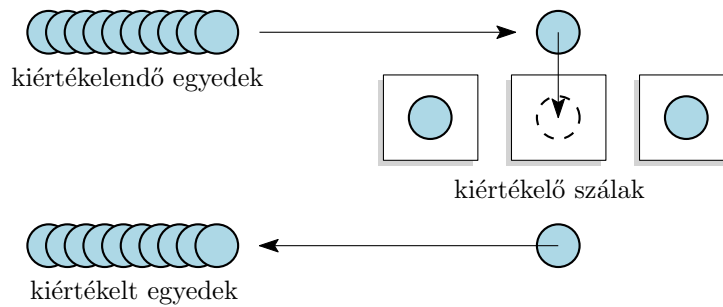
Többszálal végrehajtási modell

Több egyeden végzett bakteriális mutáció illetve génátadás műveletek egymástól függetlenül is végezhetők, akár egyidőben is. Ezt a konkurens végrehajtást több szálon egy úgynevezett *szálkészlet* végrehajtási modellen hatékonyan meg lehet tenni.

A szálkészletben előre megadott darabszámú *dolgozó szál* várja az elvégzendő feladatokat, jelen esetben egyedek hibájának kiszámítását. A kiértékelendő egyedek egy sorban helyezkednek el, és amint felszabadul egy kiértékelő szál, az a soron következő egyedet kezdi el feldolgozni a 2.6. ábra szerint.

A szálkészlet segítségével a futási sebesség jelentősen, akár többszörösére is növelhető. A kiértékelő szálak optimális száma attól függ, hogy az adott platform hány szálal tud ténylegesen párhuzamosan futtatni (processzormagok/virtuális magok száma).

A konfigurációs fájlban (ld. 2.2. kód) az evaluators opciónál adható meg, hogy hány kiértékelő szál fusson. A kiértékelés CPU-n és GPU-n is történhet, az alábbiakban utóbbi kerül ismertetésre.



2.6. ábra. A szálkészlet végrehajtási modell alkalmazása az egyedek kiértékelésére.

Hardveres gyorsítás OpenCL segítségével

A hibaszámítás a többszörös végrehajtás ellenére is számításigényes feladat, hiszen egy egyeden belül is sok tanítási mintán kell elvégezni a kiértékelést. Itt ugyanazon műveletsorozatokot kell különböző adatokkal végrehajtani, és az ilyen feladatokat grafikus processzoron (GPU-n) jellemzően nagyságrendekkel gyorsabban el lehet végezni, mint CPU-n.

Az elkészített program GPU-s implemetációt is tartalmaz az egyedek kiértékelésére, amely az OpenCL nyelvre épül, így széles körben támogat videokártyákat és egyéb számítási eszközöket. A konfigurációs fájlban (ld. 2.2. kód) megadhatók GPU-s kiértékelők, akár több különböző eszköz is egyszerre igénybe vehető. Ezt a `OpenCL_platform` és `OpenCL_device` indexekkel lehet megadni, így több videokártyával felszerelt számítógépen is kihasználható a rendelkezésre álló számítási kapacitás, ami jó skálázhatóságot jelent.

3. Mechatronikai alkalmazás

Az előző fejezetben bemutatásra került a bakteriális evolúciós algoritmus és annak alkalmazási lehetősége szimbolikus regressziós problémák esetén, valamint ennek egy saját készítésű implementációja. Ez a fejezet a továbbiakban azt ismerteti, hogy miként alkalmazhatók a bemutatott módszerek mechatronikai rendszerek modellezésére és ez milyen előnyökkel jár klasszikus megközelítésekkel szemben.

3.1. Autonóm dinamikai rendszerek modellezése

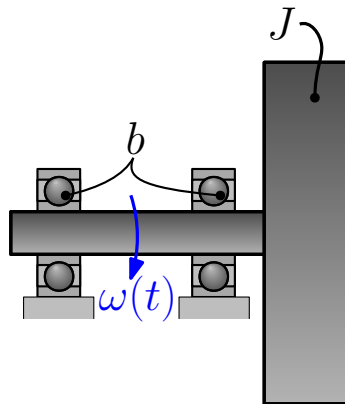
Számos mérnöki modellezési feladat visszavezethető közönséges, autonóm differenciálegyenletekre. Ebben az esetben gerjesztés nélküli, magára hagyott rendszerekről van szó, melyek dinamikája az alábbi alakú

$$\frac{d^n f(x)}{dx^n} = g\left(t, f(x), \frac{df(x)}{dx}, \frac{d^2 f(x)}{dx^2}, \dots, \frac{d^{n-1} f(x)}{dx^{n-1}}\right) \quad (3.1)$$

n -edrendű differenciálegyenlettel adható meg explicit alakban, ahol x a független változó és $f(t)$ a keresett függvény. Amennyiben a Lipschitz-feltétel teljesül, n darab kezdeti feltétel megkötésével létezik megoldás $f(x)$ -re, és az egyértelmű [24].

A továbbiakban a cél az $f(x)$ függvény bakteriális evolúciós algoritmussal történő meghatározása úgy, hogy a tanítási adatok az adott differenciálegyenlet bizonyos partikuláris megoldásaiból vett minták legyenek.

Tekintsük a 3.1. ábrán látható forgó mechanikai rendszert, ami a J tehetetlenségi nyomatékú forgórészből és b viszkózus csillapítási tényezőjű csapágyszárazságból áll.



3.1. ábra. A viszkózan csillapított forgórész mechanikai modellje.

A forgórész $\omega(t)$ szögsebességére felírható a

$$J\dot{\omega}(t) = -b\dot{\omega}(t) \quad (3.2)$$

mozgásegyenlet, ami megfelel az

$$\dot{f}(t) - \lambda f(t) = 0 \quad (3.3)$$

elsőrendű, lineáris, állandó együtthatós homogén differenciálegyenletnek, ahol $f = \omega$ és a differenciálegyenlet karakterisztikus gyöke $\lambda = -b/m$. Ennek megoldása $f(0) = f_0$ kezdeti feltételre zárt alakban, analitikusan meghatározható az alábbi összefüggés szerint:

$$f(t) = f_0 e^{\lambda t} \quad (3.4)$$

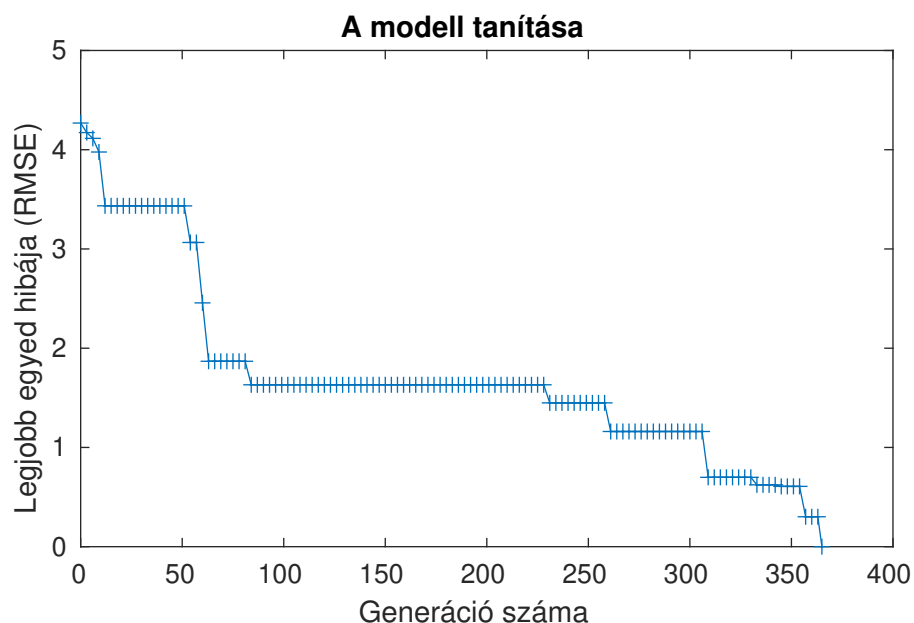
A továbbiakban vizsgálatra kerül, hogy a tárgyalt autonóm dinamikai rendszer viselkedése miként modellezhető bakteriális evolúciós algoritmus segítségével. Ebben az esetben a szimbolikus regressziós feladat célja az $f(t)$ függvény meghatározása. A módszer általánosítási képességének tesztelése végett nem csupán a t időpillanat lesz a keresett függvény argumentuma, hanem szimbolikusan (változóként) szerepel a λ karakterisztikus gyök, mint a rendszer paramétere, és az f_0 kezdeti érték.

Legyenek a tanítási adatok a differenciálegyenlet partikuláris megoldásaiból vett minták, különböző paraméterek és kezdeti feltételek esetén. A tanítási fájl részletét a 3.1. kód mutatja.

A tanítási folyamat során a legjobb egyed hibáját a 3.2. ábra mutatja. A hiba néhány száz generáció után megszűnt, a végeredményként kapott legjobb egyedek a 3.2. kódban láthatók prefix kifejezés alakban.

t	lambda	f0	
0	-5	-1	-1
0	-5	-0.8	-0.8
0	-5	-0.6	-0.6
...			

3.1. kód. A vizsgált autonóm dinamikai rendszerhez tartozó tanítási adatok.



3.2. ábra. A legjobb egyed hibája a tanítási folyamat során.

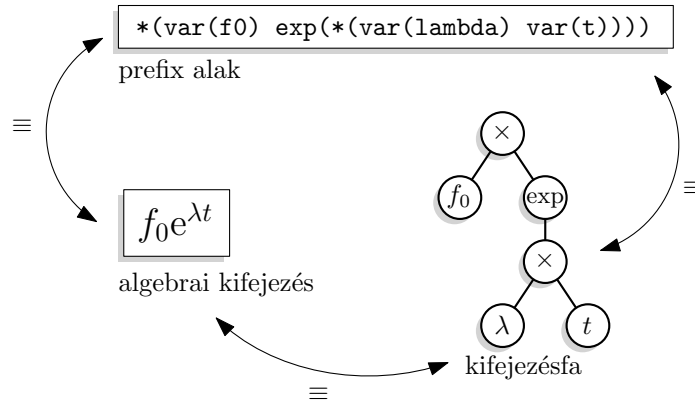
```

*(var(f0) exp(*(var(lambda) var(t))))
-(*(var(f0) exp(*(var(lambda) var(t)))) const(1))
  sgn(var(t)))
*(var(f0) *(var(t) exp(-(var(lambda) cos(*(exp(var(t)
  ) const(-1.26831)))))))
*(var(t) *(+(const(0.000182909) *(sin(var(t)) exp(var
  (lambda)))) var(f0)))
+(*(var(f0) *(var(t) exp(-(var(lambda) cos(*(exp(var(
  t)) const(4.80636))))))) var(f0))
*(var(t) *(+(sin(var(lambda)) *(var(t) exp(var(lambda)
  )))) var(f0)))
...

```

3.2. kód. A vizsgált rendszer modellezésének eredményeként kapott populációs fájl.

A legjobb egyedet 3.3. ábra szemlélteti a populációs fájlban szereplő prefix alakban, valamint ezzel ekvivalens módon kifejezésfával és algebrai kifejezésként reprezentálva. Utóbbi teljesen megegyezik a 3.4. egyenlettel, így megállapítható, hogy a bakteriális evolúciós algoritmus képes volt kizárólag partikuláris megoldások ismeretében előállítani a vizsgált autonóm differenciálegyenlet analitikus, általános megoldását.



3.3. ábra. A tanítás eredményeként kapott legjobb egyed különböző reprezentációi.

3.2. Rendszeridentifikáció

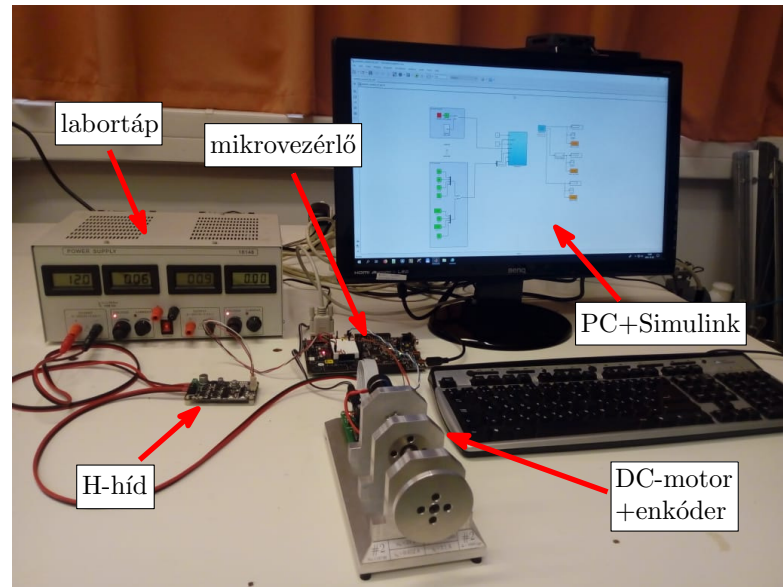
A mechatronika egyik alapvető feladata a rendszeridentifikáció, amelynek célja az, hogy mérési adatok alapján meghatározható legyen egy adott rendszer dinamikai modellje. Amennyiben a rendszert leíró egyenletek ismertek, de ennek együtthatói és egyéb paraméterei ismeretlenek, úgy paraméteridentifikációról vagy paraméterbecslésről [25] van szó. Ezt más néven *szürke doboz* (angolul *grey-box*) identifikációnak is nevezik, utalván arra, hogy bizonyos információk ismertek a rendszer dinamikáját illetően.

A paraméteridentifikációs feladatok matematikailag megfogalmazhatók egy többváltozós függvény szélsőérték keresési problémájaként. Gyakran ez egy hibafüggvény minimalizálását jelenti, ami klasszikus optimalizációs eljárásokkal vagy genetikus algoritmusokkal hatékonyan megtehetőek. A paraméteridentifikáció azonban csak akkor alkalmazható, ha megfelelő előismeretek állnak rendelkezésre a vizsgált rendszer modelljének matematikai struktúrájáról.

Amennyiben a vizsgált rendszerről csak a bemenő és a kimenő jelek sorozata ismert, úgy *fekete doboz* identifikációról [25] beszélünk. Ebben az esetben nem adott számú paraméter behangolását kell elvégezni, hanem a modell struktúráját is identifikálni kell, így lényegesen összetettebb feladatról van szó. A továbbiakban egy ilyen jellegű identifikációs probléma kerül bemutatásra klasszikus megközelítésben és bakteriális evolúciós algoritmus segítségével.

3.2.1. A vizsgált rendszer

Az identifikáció során a 3.4. ábrán látható kefésegyenáramú motoros villamos hajtás modelljének meghatározása volt a feladat. A motor kapocsfeszültségét H-híd állította elő, melynek vezérlését mikrovezérlő végezte, a mérésadatgyűjtés pedig PC segítségével történt.



3.4. ábra. Az identifikáció során használt mérési elrendezés.

A rendszer bemenete a motor termináljaira kapcsolt feszültség, a kimenet pedig a forgórész szögsebessége. Közvetlen szögsebességmérésre nem volt lehetőség, így a szögsebesség inkrementális enkóderrel mért pozíciójel numerikus deriválásával lett becsülve.

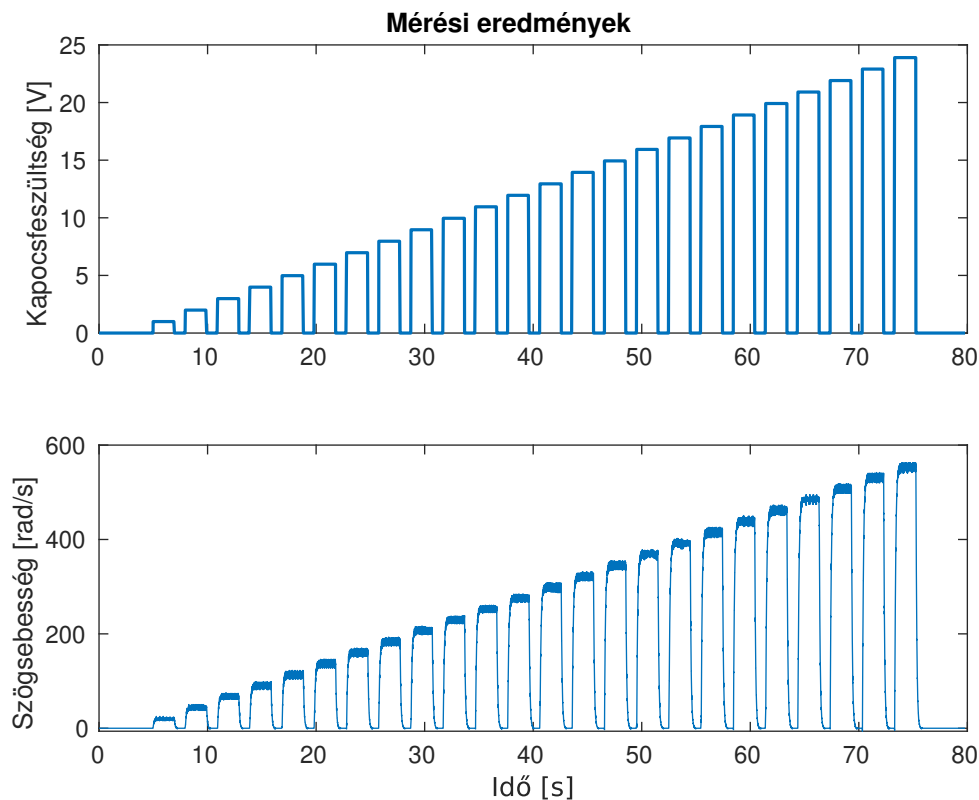
A mérés során alkalmazott vizsgálójel különböző nagyságú ugrásfüggvényekből tevődött össze. A mérési eredményeket a 3.5. ábra mutatja. Jól látható, hogy a szögsebesség zajjal terhelt, amit a szögelfordulás mérésének kvantálási hibája és a numerikus deriválás okoz.

3.2.2. Klasszikus identifikációs eljárások

A fekete doboz identifikációs feladatok klasszikus megközelítésben egy jól ismert modell kiválasztására és annak paraméteridentifikációjára vezethetők vissza. Egy ilyen, identifikációra gyakran alkalmazott modell az úgynevezett ARX (Autoregressive with Extra Input) modell, amelyet a

$$y[k] + a_1 y[k-1] + \dots + a_{n_a} y[k-n_a] = b_1 u[k-n_d] + \dots + b_{n_b} u[k-n_b-n_d] + e[k] \quad (3.5)$$

differenceciaegyenlet ír le, ahol $y[k]$ a rendszer kimenete a k -adik időpillanatban, $u[k]$ a bemenő jel és $e[k]$ a mérésből származó zaj. Az n_a és n_b paraméter megmutatja, hogy a kimenet kiszámításához hány korábbi időlépés kimeneti illetve bemeneti értékei kerültek felhasználásra. Az n_d paraméter időkésést tartalmazó rendszerek esetén azt mutatja meg, hogy hány időlépésnek kell eltelnie addig, amíg a bemenőjel hatása a kimeneten megjelenik.



3.5. ábra. Az identifikált rendszerre kapcsolt vizsgálójel és a rendszer válasza.

A vizsgált villamos hajtás esetén az identifikáció a MATLAB System Identification Toolbox segítségével került elvégzésre $n_a = 2$, $n_b = 3$ és $n_d = 0$ paraméterekkel. Az identifikáció eredményét a 3.1. táblázat mutatja.

3.1. táblázat. Az identifikált ARX modell paraméterei.

paraméter	érték
a_1	-0,3598
a_2	-0,6201
b_1	0,3407
b_2	-0,0001337
b_3	0,1233

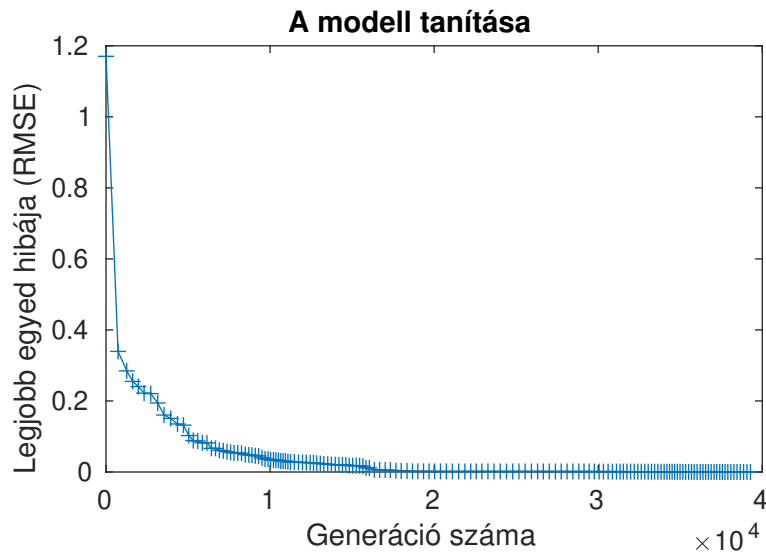
3.2.3. Indentifikáció bakteriális evolúcióval

A klasszikus identifikációs eljárás pontossága nagyban függ az alkalmazott modell típusától, valamint annak előzetesen megválasztott paramétereitől (az előző példában $n_{a,b,d}$). Az identifikáció során így a struktúra adott, és csak a paraméterek hangolása történik, ami egy alapvető korlátja az alkalmazásnak. Bakteriális evolúciós algoritmus segítségével azonban az egyedek struktúrája dinamikusan változik, ami fekete doboz identifikáció során kihasználható. Keressük a modellt az

$$y[k] = f(y[k-1], y[k-2], u[k], u[k-1], u[k-2]) \quad (3.6)$$

alakban, ami tekinthető a 3.5. egyenlet általánosításának, ugyanis itt nem tételezünk fel lineáris kapcsolatot a múltbéli bemenetek és kimenetek között. Az f függvény meghatározása így szimbolikus regressziós feladatként értelmezhető, amely a 2. fejezetben ismertetett módon végezhető el.

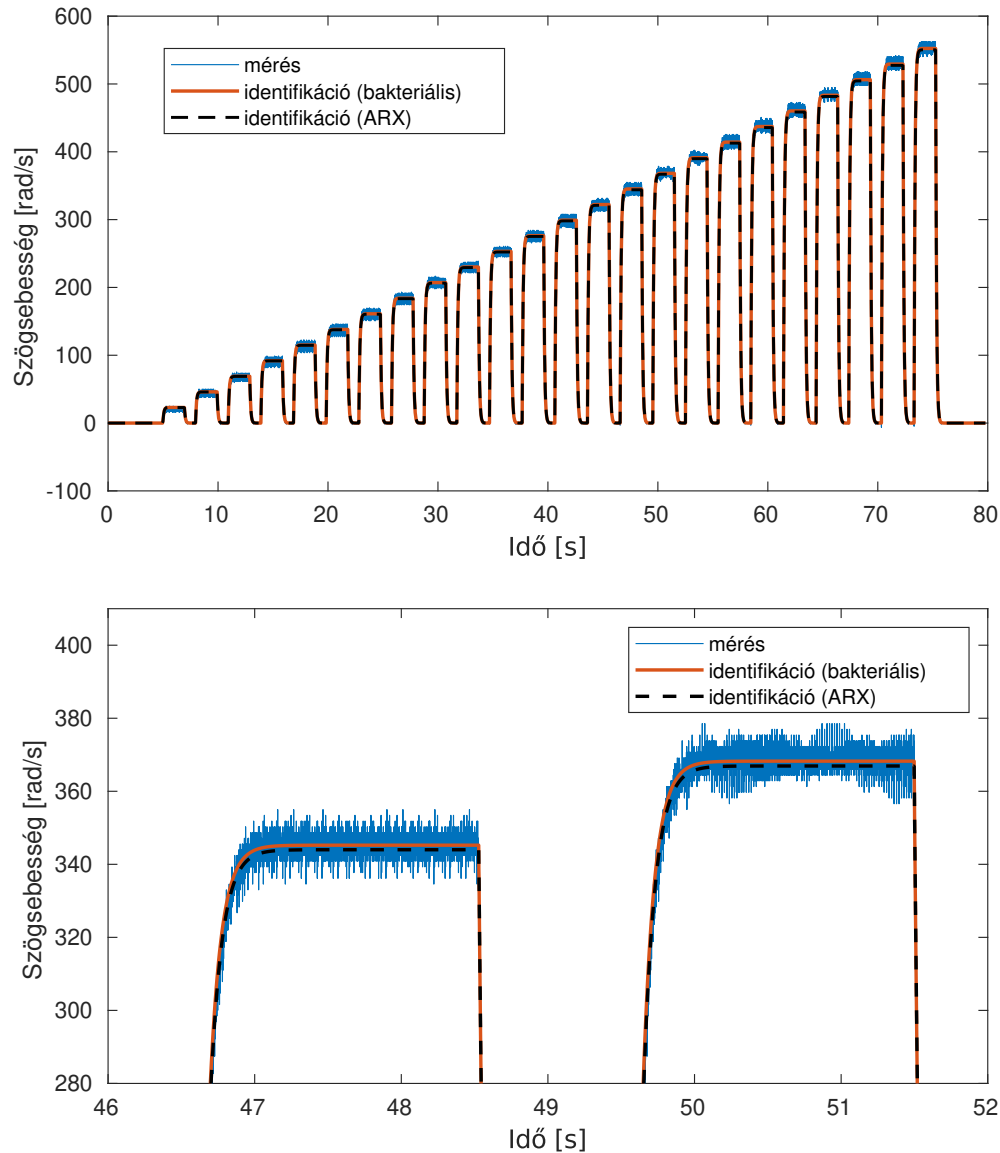
A bakteriális evolúciós algoritmus alkalmazásához a mérésből származó adatok tanító mintákként kerültek felhasználásra. A tanítási folyamatot a 3.6. ábra mutatja, a kapott legjobb egyed a 3.3. kódrészletben látható. Az eredményként előállt algebrai kifejezés egy olyan diszkrét idejű modell, amit a 3.6. egyenletbe visszahelyettesítve szimuláció végezhető. Az így kapott identifikált rendszer válaszát a 3.7. ábra mutatja.



3.6. ábra. Az identifikáció legjobb egyedének hibája a különböző generációk során.

Jól látható, hogy mind a klasszikus, mind az új megközelítésben kapott modell pontosan követi a valós rendszer viselkedését a mérést terhelő zajok ellenére. A bakteriális evolúciós megközelítés azonban nem egy adott struktúrájú modelltől indult ki, hanem magát a struktúrát is képeses identifikálni.

Ez lehetővé teszi a különböző nehezen modellezhető, nemlineáris hatás pontosabb közelítésére. A 3.7. ábra alsó részén látható, hogy az ARX modell a szögsebességet kissé alábecsli a bakteriális evolúcióval előállított modellel szemben. Ez egy lehetséges következménye annak, hogy a teljes működési tartományon egy lineáris modell kevésbé tudja pontosan modellezni a valós rendszer működését, szemben a modell struktúráját és bonyolultságát nem korlátozó bakteriális evolúciós megközelítéssel.



3.7. ábra. A mérési adatok és a különböző módon identifikált modellek összehasonlítása.

```

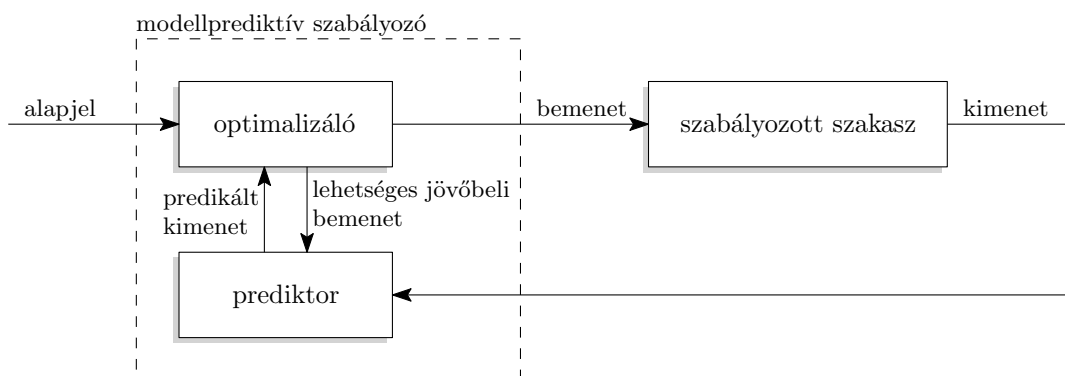
+(var(y_2) /(-(/(*(const(-3.96466) var(u_2)) const
(-2.26075)) *(const(5.30931) -(-(var(y_2) +(var(y_1)
) /(-(var(y_1) var(y_2)) cos(sgn(var(y_2)))))) *
const(-0.0242587) *(const(-6.46352) -(*(var(u_0)
const(-4.10504)) const(-5.11704)) var(y_1))))))
const(40.2537)))

```

3.3. kód. Az identifikáció eredményeként kapott legjobb egyed prefix alakban.

3.3. Modellprediktív szabályozás

A modellprediktív szabályozás során a szabályozó a prediktor segítségével becslést ad különböző beavatkozások esetén szabályozott folyamat viselkedésére, és ez alapján dönt a ténylegesen alkalmazott beavatkozásról. Egy ilyen szabályozási lánc felépítését mutatja a 3.8. ábra.



3.8. ábra. A modellprediktív szabályozási lánc elvi felépítése.

A *prediktor* a szabályozott szakaszt modellezi, és annak korábbi be- és kimenetei, valamint jövőbeli lehetséges bemenetei alapján képes megbecsülni a jövőbeli kimeneteket. A szabályozó alapjele és a predikciók alapján az *optimalizáló* egység eldönti, hogy melyik forgatókönyv a legkedvezőbb, és annak megfelelő jellel hat a szabályozott szakaszra. Az optimalizálás rendszerint valamilyen hibafüggvény alapján történik, amivel a beállási jellemzők és a szabályozásra vonatkozó kényszerek is megadhatók.

Az előzőekben bemutatásra került, hogy bakteriális evolúciós algoritmussal dinamikai rendszerek modellezése és rendszeridentifikáció is megvalósítható. Ezen eredmények alapján felmerülhet lehetséges alkalmazási területként is egy olyan modellprediktív szabályozó, amelyben a prediktort bakteriális evolúciós algoritmus állítja elő. A továbbiakban egy ilyen szabályozó megvalósítása kerül bemutatásra szimulációs eszközökkel.

3.3.1. A szabályozott szakasz modellje

Vizsgáljuk a 3.9 ábrán látható, villamos motoros hajtás modelljét. A modell villamos alrendszerében az R_a armatúra ellenállása, az L_a induktivitás, k_m és k_e a nyomaték- és szögsebességállandók. A meghajtott $\omega(t)$ szögsebességű forgórész J tehetetlenségi nyomatékú és a csapágyazás b_{cs} viszkózus csillapítással, valamint M_C Coulomb-súrlódási nyomatékkal jellemezhető. Az armatúra $i(t)$ áramát szeretnénk szabályozni a motor termináljaira kapcsolt $u_0(t)$ feszültséggel. A paraméterek numerikus értékeit a 3.2. táblázat tartalmazza.

3.2. táblázat. A szimulált rendszer paraméterei.

paraméter	érték	mértékegység
R_a	5	Ω
L_a	5	mH
k_m, k_e	0,03	$\frac{\text{Nm}}{\text{A}}$
J	$2 \cdot 10^{-5}$	kgm^2
b_{cs}	$2,5 \cdot 10^{-5}$	$\frac{\text{Nm}}{\text{rad}}$
M_C	$6 \cdot 10^{-4}$	Nm
u_{dc}	12	V
T_s	0,05	ms

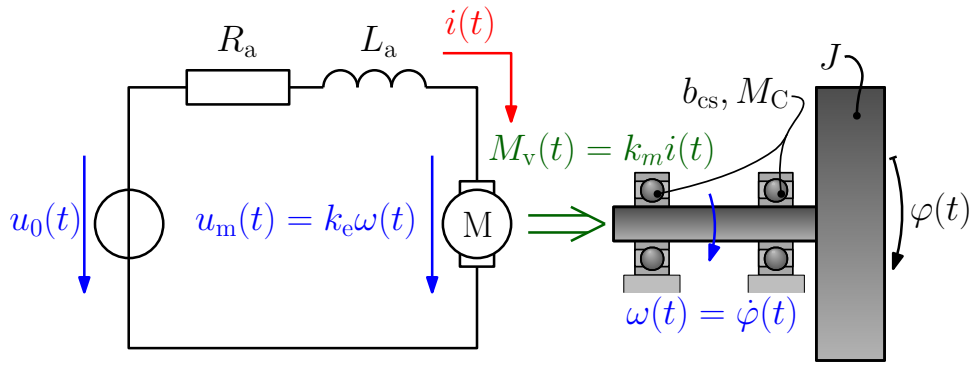
A motor dinamikáját a villamos oldalra felírható

$$u_0 = R_a i(t) + L_a \frac{di(t)}{dt} + k_e \omega(t) \quad (3.7)$$

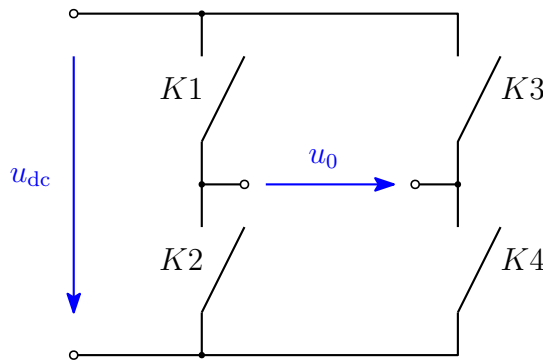
hurokegyenlet és a mechanikai oldalra felírható

$$J \frac{d\omega(t)}{dt} = k_m i(t) - b_{cs} \omega(t) - M_C \cdot \text{sgn}(\omega(t)) \quad (3.8)$$

mozgásegyenlet írja le. A motor meghajtását a 3.10. ábrán látható H-híd végzi, melyben u_{dc} a buszfeszültség és az egyes hídágakban található félvezetők ideális kapcsolóként vannak modellezve. Az alkalmazott kapcsolási állapotokat a 3.3. táblázat mutatja.



3.9. ábra. A szabályozott villamos motoros hajtás modellje.



3.10. ábra. A H-híd modellje.

3.3. táblázat. A H-híd vezérlése során alkalmazott kapcsolások.

K1	K2	K3	K4	u_0
vezet	nem vezet	nem vezet	vezet	u_{dc}
vezet	nem vezet	vezet	nem vezet	0 V
nem vezet	vezet	nem vezet	vezet	0 V
nem vezet	vezet	vezet	nem vezet	$-u_{dc}$

3.3.2. Prediktor tervezése klasszikus megközelítés alapján

A prediktor feladata, hogy korábbi mérések adatai alapján az adott beavatkozáshoz előállítsa a jövőbeli kimenet becsült értékét. Mivel maga a modellprediktív szabályozó és a prediktor is diszkrét időben működő rendszer, így be kell vezetni a T_s mintavételi időt, valamint a diszkrét idejű jeleket. Jelölje

$$u[k] = u_0(kT_s) \quad (3.9)$$

a k -adik időlépés kapocsfeszültségét, valamint

$$i[k] = i(kT_s) \quad (3.10)$$

a k -adik időlépés armatúra áramát. A prediktor egy olyan

$$i[k+1] = f(i[k], i[k-1], \dots, u[k], u[k-1], \dots) \quad (3.11)$$

alakú függvény, ami a k -adik időpillanatban becslést ad a következő időlépés $i[k+1]$ áramára, amennyiben $u[k]$ bemeneti feszültség hat a rendszerre. Ehhez felhasználja az adott aktuális és a korábbi időlépések méréseit. A 3.7. egyenlet diszkretizálásával

$$u[k] = R_a i[k] + L_a \frac{i[k+1] - i[k]}{T_s} + \underbrace{k_e \omega(kT_s)}_{u_i} \quad (3.12)$$

adódik, ahol a differenciálhányadost előrelépő Euler-módszer közelíti. Jelölje a továbbiakban u_i a motor indukált feszültségét, és tegyük fel, hogy a predikció időtartama alatt ez állandó értékű. Ez a feltételezés azért tehető meg, mert a mechanikai tranziens jelenségek jellemzően nagyságrendekkel lassabbak a villamos alrendszerénél [26]. Az egyenlet átrendezhető

$$i[k+1] = i[k] + \frac{T_s}{L_a} (u[k] - R_a i[k] - u_i) \quad (3.13)$$

alakra, ami már explicit módon kifejezi a predikált áramot, azonban ehhez az u_i értéket is ki kell számítani. Mivel u_i értéke közelítőleg állandó, így kifejezhető 3.12. egyenlet időben eltolat alakjából

$$u_i = u[k-1] - R_a i[k-1] - L_a \frac{i[k] - i[k-1]}{T_s} \quad (3.14)$$

módon. Ezzel a prediktor klasszikus megközelítésben elkészült, a későbbiekben bemutatásra kerül ennek alkalmazása a modellprediktív szabályozásban.

A modell előállítása során két fontos közelítés/elhanyagolás történt: az áram deriváltját az előrelépő Euler-módszer közelítette, valamint a teljes mechanikai alrendszer viselkedését egy időben állandónak tekintett indukált feszültség jellemezte.

Egy ennél is pontosabb modell előállításához a 3.7. differenciálegyenlet megoldását kellene jobban közelíteni, valamint a mechanikai alrendszer viselkedését is figyelembe kellene venni. Ez azonban a tervezést nehézkessé teszi, így a továbbiakban ezt feladatot a bakteriális evolúciós algoritmus fogja ellátni.

3.3.3. Prediktor tervezése bakteriális evolúciós algoritmussal

A prediktor, ahogy azt általános formában a 3.11. egyenlet mutatja, egy többváltozós függvény, melynek argumentumai az aktuális időlépés mért kimenete, lehetséges beavatkozása, valamint a múltbéli mért kimenetek és tényleges beavatkozások. Ennek a függvénynek a meghatározása szimbolikus regressziós feladatként értelmezhető, így bakteriális evolúciós algoritmussal is előállítható.

A klasszikus megközelítésen alapuló prediktorhoz hasonlóan itt is felhasználásra kerülnek egy időlépéssel korábbi adatok, így változóként $i[k-1], i[k], u[k-1], u[k]$ szerepelnek. Ezeket rendre $i1, i0, u1$ és $u0$ jelöli a tanítási fájlban, ahogy azt a 3.4. kód is mutatja.

i1	i0	u1	u0	
-5	-4.78536	-12	-12	-4.58109
-4	-3.83414	-12	-12	-3.67629
-3	-2.88292	-12	-12	-2.77149
-2	-1.9317	-12	-12	-1.8667
-1	-0.980484	-12	-12	-0.961904
0	-0.0292652	-12	-12	-0.0571084
1	0.921953	-12	-12	0.847687
...				

3.4. kód. A prediktor tanításához generált tanítási adatok.

A tanítási adatok előállítása szimuláció segítségével történt a 3.7 és a 3.8. egyenletek numerikus megoldásával. Tekintsük a $t = 0$ időpillanatot a predikció kezdetének. Mivel a prediktor egy mintavétellel korábbi adatokat is felhasználhat, így az egyes tanítási minták előállításához szükséges kezdeti feltételek a $t = -T_s$ időpillanattól indulnak ki. Így a kiindulási adatok az

$$i(-T_s) = i[k-1] \quad (3.15)$$

armatúra áramra és a

$$\omega(-T_s) = \omega[k-1] \quad (3.16)$$

szögsebességre vonatkozó kezdeti feltételből, valamint az

$$u(t) = \begin{cases} u[k-1] & \text{ha } t < 0 \\ u[k] & \text{ha } t \geq 0 \end{cases} \quad (3.17)$$

szakaszonként állandó bemeneti feszültségfüggvényből álltak. A rendszer dinamikáját leíró differenciálegyenletrendszer numerikus megoldása alapján meghatározható

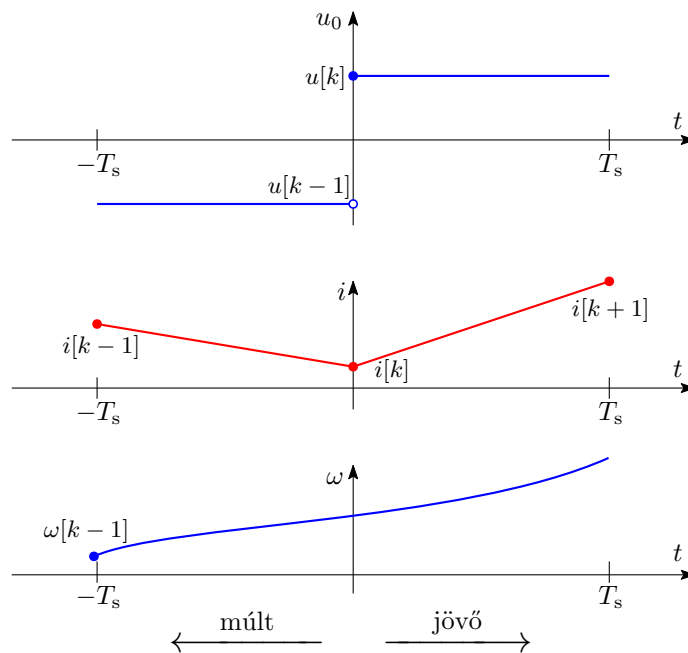
a predikció időpillanatában mért

$$i[k] = i(0) \quad (3.18)$$

áram és a jövőbeli

$$i[k+1] = i(T_s) \quad (3.19)$$

áram, ami a prediktor elvárt kimenete. Egy adott tanítási minta előállítása során kialakuló időfüggvényeket a 3.11. ábra szemlélteti. A szimuláció paraméterei úgy lettek megválasztva, hogy széles működési tartományt fedjenek le. A 3.4. táblázat mutatja külön-külön az egyes paraméterek lehetséges értékeit, a tanítási adatsor ezek összes lehetséges kombinációjából került generálásra.

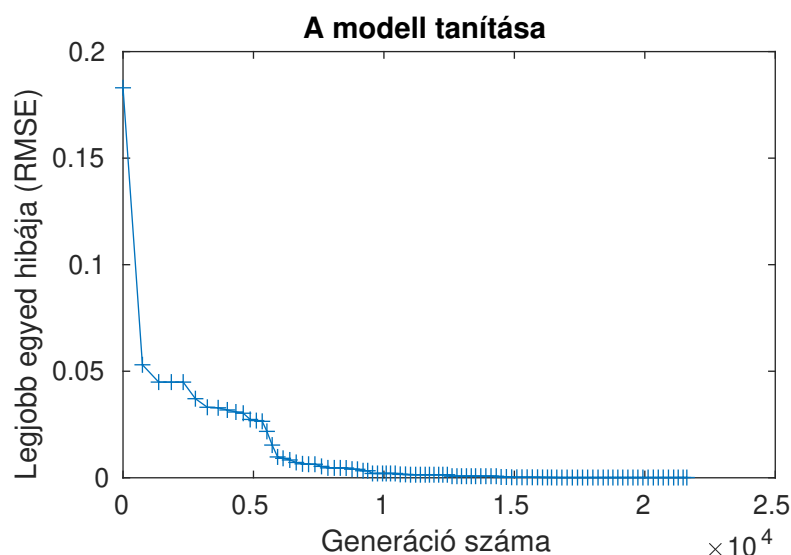


3.11. ábra. A prediktor tanításához szükséges szimuláció időfüggvényei és az ezekből vett minták.

3.4. táblázat. A prediktor tanítási mintáinak generálása során alkalmazott paraméterek.

paraméter	lehetséges értékek	mértékegység
$i[k-1]$	$\{-5, \dots, -1, 0, 1, \dots, 5\}$	A
$\omega[k-1]$	$\{-300, \dots, -60, 0, 60, \dots, 300\}$	$\frac{\text{rad}}{\text{s}}$
$u[k-1]$	$\{-u_{\text{dc}}, 0, u_{\text{dc}}\}$	V
$u[k]$	$\{-u_{\text{dc}}, 0, u_{\text{dc}}\}$	V

A bakteriális evolúciós algoritmus futtatási folyamatát a 3.12. ábra mutatja. Az egyedek hibája közvetlenül megmutatja az áram predikciójának pontosságát. Látható, hogy a tanítási folyamat során a hiba néhány ezer generáció után közelítőleg nullára csökken. Jelen esetben a 21610. generációban a legjobb egyed hibája 0,000117164, ami predikciós hibaként is értelmezhető amperben.



3.12. ábra. A prediktor hibája a bakteriális evolúció során.

A prediktorként előállított legjobb egyed prefix alakban a 3.5. kódban látható. Bár ezen algebrai kifejezés komplexitása jóval meghaladja a klasszikus megközelítésben kapott prediktort (3.13 és 3.14 egyenletek), mégis kezelhető méretű maradt ahhoz, hogy valós időben mikrovezérlőn vagy DSP-n implementálható legyen.

```
-(var(i0) /((- (var(i0) /(+ (var(i1) +(+ (const
  (4.93637) +(+ (+ (+ (+ (var(u0) var(i1)) var(i1)) *(
    const(-0.412279) -(- (var(u1) const(0.000430667))
    sgn(/(- (var(u1) const(-0.218899)) const(-0.412122))
  )))) /(- (var(i0) var(i1)) const(0.0305224))) *(
    const(-0.478835) + (var(u1) -(- (sin(sgn(+ (const
      (-8.77194) var(u1)))) + (+ (/ (var(i1) const(2.18225))
      var(i1)) var(i0))) + (sgn(- (* (const(5.14389) sin(
        var(u1))) var(i1)) + (var(u1) var(u1)))) var(i1))))
  ) /(- (var(i0) var(i1)) const(0.0277718))) const
    (-4.93576)) /(- (var(i0) var(i1)) const(0.0305224)))
  ) const(4.66231))) const(9.87548)) const(2.2251)))
```

3.5. kód. A prediktor tanítása eredményeként kapott legjobb egyed.

3.3.4. A prediktív szabályozás szimulációja

A továbbiakban a bemutatott prediktorok modellprediktív áramszabályozás szimulációjával kerülnek összehasonlításra, mely során a cél az armatúra áramának i_{ref} referenciaértéken tartása. A szabályozási algoritmus az alábbi lépésekből áll:

1. Az aktuális $i[k]$ áram mérése.
2. Az $i[k + 1]$ áram predikciójának kiszámítása $u[k]$ lehetséges értékei alapján.
 - (a) $u[k] = -u_{dc}$ esetén legyen $i^{-}[k + 1]$ a predikált áram
 - (b) $u[k] = 0$ V esetén legyen $i^0[k + 1]$ a predikált áram
 - (c) $u[k] = u_{dc}$ esetén legyen $i^{+}[k + 1]$ a predikált áram
3. Az $e(i_{ref}, i[k + 1])$ hibafüggvény kiértékelése minden predikcióra.
4. Az $u[k]$ értéke legyen az a választás, amelyik a legkisebb hibát eredményezte.
5. A beavatkozó jel legyen állandó $u(t) = u[k]$ értékű a következő mintavételi időpillanatig.
6. Ismétlés az 1. lépéstől.

A 2. lépésben található predikció egyaránt elvégezhető a klasszikus megközelítésben és a bakteriális evolúciós algoritmussal kapott prediktor segítségével is. A 3. lépésben található hibafüggvény az egyszerűség kedvéért

$$e(i_{ref}, i[k + 1]) = (i_{ref} - i[k + 1])^2 \quad (3.20)$$

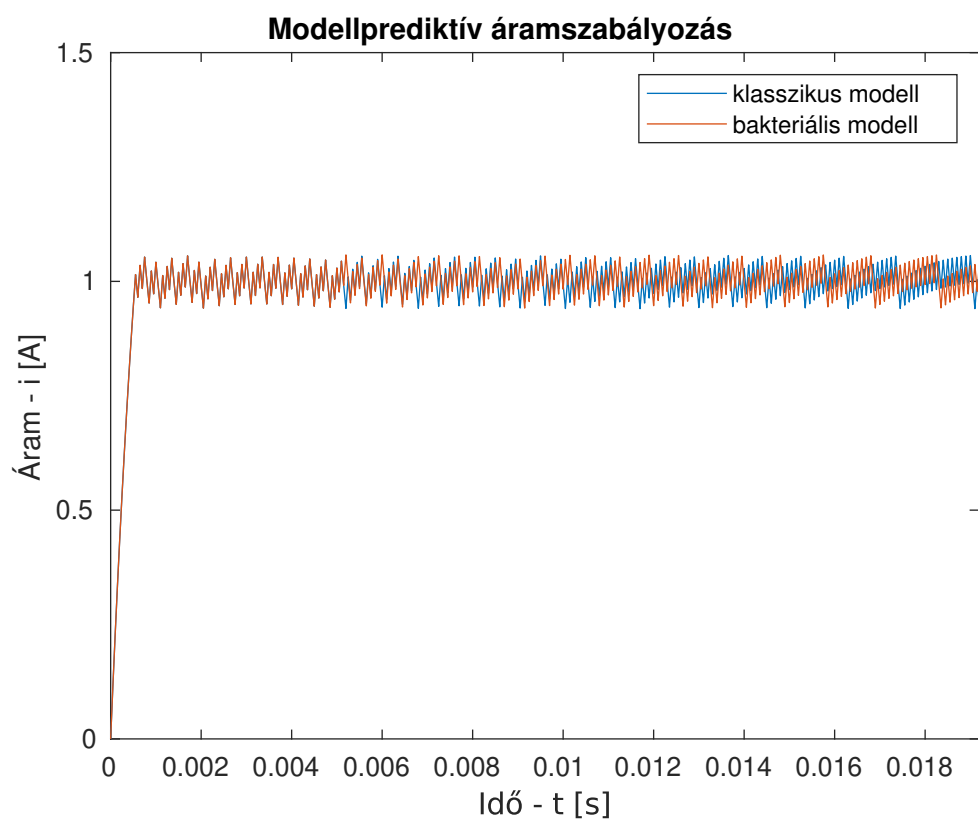
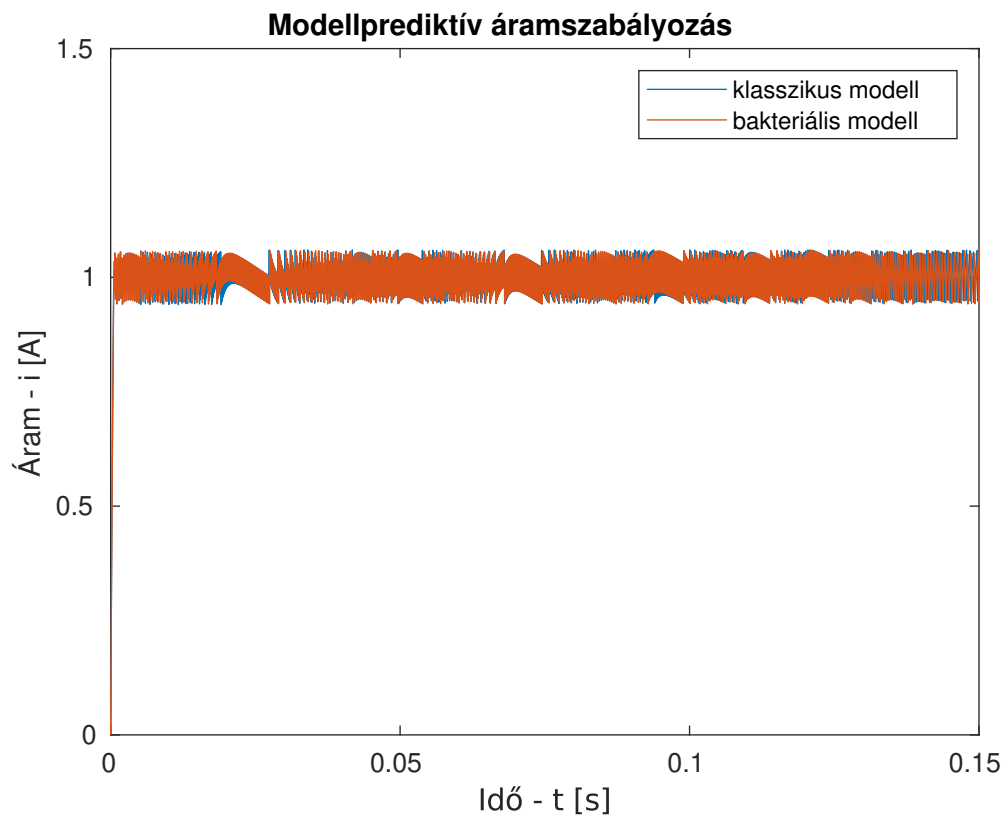
módon csak a referenciától való négyzetes eltérés nagyságával arányos. A rendszer dinamikája a 3.7 és a 3.8 differenciálegyenletek numerikus megoldásával

$$i(0) = 0 \text{ A} \quad \text{és} \quad \omega(0) = 0 \frac{\text{rad}}{\text{s}} \quad (3.21)$$

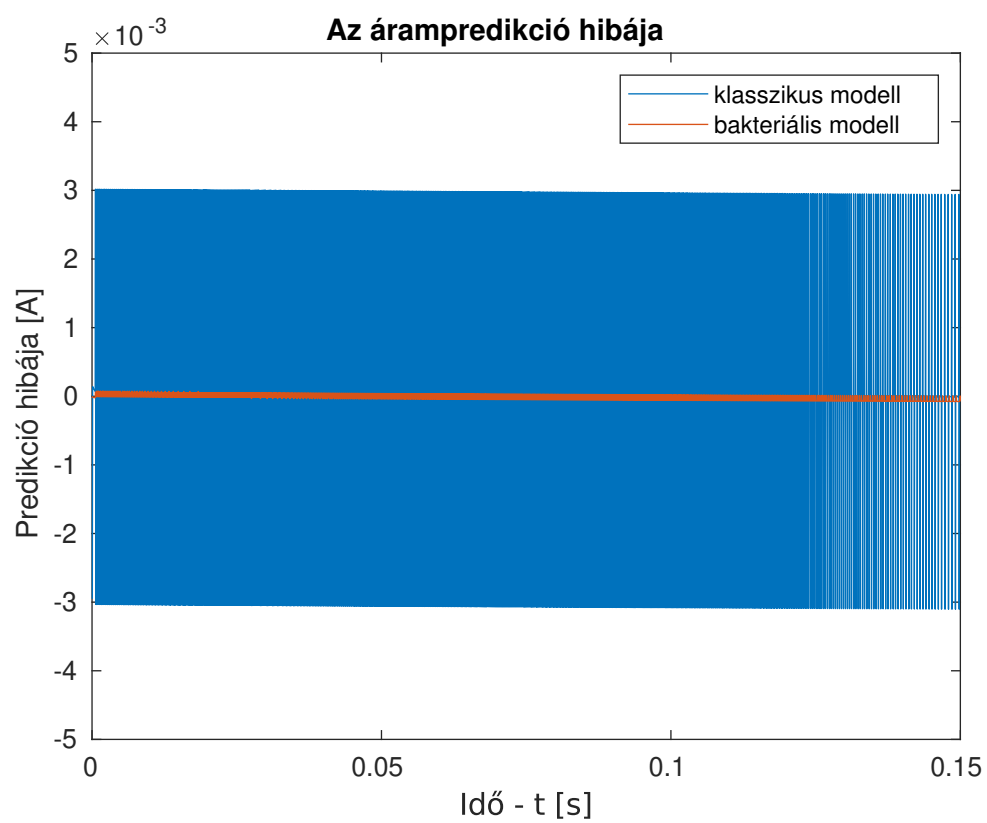
kezdeti feltételekből kiindulva került meghatározásra.

Az armatúra áram időfüggvénye a 3.13. ábrán látható. Megállapítható, hogy mindkét prediktor alkalmazása során hasonló áram jelalakok alakulnak ki, amelyek túllövés nélkül, minimális beállási idő alatt érik el a referenciának beállított 1 A értéket.

A predikált áram és a következő időpillanatban mért tényleges áram különbségéből képezhető a predikciós hiba, ami a 3.14. ábrán látható. A klasszikus prediktor hibája közelítőleg ± 3 mA között ingadozik, míg ezzel szemben a bakteriális evolúcióval előállított prediktor alkalmazása legrosszabb esetben is csak 0,075 mA hibát eredményezett, ami nagyságrendi eltérés a klasszikus módszerrel szemben.



3.13. ábra. A modellprediktív módon szabályozott armatúra áram.



3.14. ábra. Az áram predikciójának hibája különböző predikciós módszerekkel.

4. Összefoglalás

Ezen dolgozat a bakteriális evolúciós algoritmusok mechatronikai alkalmazási lehetőségeit tárgyalta. Bemutatásra került a szimbolikus regressziós probléma, amely során algebrai kifejezések optimalizálása a cél, valamint ennek egy lehetséges megvalósítása bakteriális evolúciós algoritmussal. Ezen felül példákon keresztül bemutatásra került az algoritmus alkalmazása autonóm dinamikai rendszerek modellezése, rendszeridentifikáció és végül modellprediktív szabályozás.

Elkészült a bakteriális evolúciós algoritmus egy saját készítésű implementációja C++ nyelven. A program nem tartalmaz platformspecifikus kódrészletet, így platformfüggetlen, Windows és Linux rendszereken egyaránt tesztelésre gm került. A hatékony végrehajtás érdekében több szálon, vagy akár GPU-s gyorsítással is futtatható a bakteriális evolúciós algoritmus, így könnyen skálázható nagyobb számítási kapacitással bíró platformokra.

A dolgozatban elsőként autonóm dinamikai rendszerek modellezésén keresztül került bemutatásra a bakteriális evolúciós algoritmus. A módszer képes hatékonyan előállítani a vizsgált dinamikai rendszer egzakt megoldását explicit, zárt alakban úgy, hogy a tanítás során kizárólag partikuláris megoldásokból vett minták voltak elérhetők. A módszer nagy előnye, hogy a megoldásban nem csak a független változó szerepelhet szimbolikusan, hanem tetszőleges paraméterek és kezdeti feltételek is.

A bemutatott módszer adott típusú differenciálegyenletek teljes osztályát képes megoldani, szemben a numerikus iteratív eljárásokkal, melyek csak adott konkrét paraméterekhez és kezdeti feltételekhez tartozó partikuláris megoldást közelítik. Amennyiben az adott problémához létezik analitikus megoldás, úgy a módszer képes azt is meghatározni, ellenkező esetben közelítő megoldást ad.

Ezen felül kísérleti eszközök segítségével bemutatásra került, hogy bakteriális evolúcióval valós fizikai rendszerek identifikációja is elvégezhető. Klasszikus identifikációs eljárásokkal szemben nem szükséges előzetes ismeretekkel rendelkezni a vizsgált rendszer struktúráját illetően, mert a bakteriális evolúció során az egyedek struktúrája is dinamikusán változik. Ez lehetővé teszi a nehezen modellezhető, nemlineáris hatásokat, például a száraz sűrűdést tartalmazó rendszerek identifikációját is, mint amilyen a bemutatott villamos motoros hajtás.

Végül bemutatásra került szimulációs eszközökkel egy villamos motoros hajtás modellprediktív áramszabályozása. A prediktor klasszikus módon és bakteriális evolúciós algoritmus segítségével is előállítható. Utóbbi módszer pontosabbnak bizonyult, ugyanis az identifikációhoz hasonlóan azok a jelenségek is figyelembe vehetők, amiket a klasszikus modellek elhanyagolnak.

Bár a különféle gépi tanulási módszerek a mérnöki élet sok területén elterjedtek, a bakteriális evolúciós algoritmusok alkalmazása kevésbé tárgyalt téma a mechatronikában. Ezen dolgozat eredményei alapján azonban látható, hogy ez a jó adaptációs képességgel bíró módszer, amely képes összetett rendszerek modellezésére, és jól használható mind a rendszertechnika, mind pedig az irányítástechnika területén egyaránt.

5. Felhasznált irodalom

- [1] Budai, C., Kovács, L. L., Kövecses, J., Stépán, G., „Combined effects of sampling and dry friction on position control”, *Nonlinear Dynamics*, 98(4), pp. 3001–3007, 2019. doi:[10.1007/s11071-019-05101-7](https://doi.org/10.1007/s11071-019-05101-7)
- [2] Astrom, K. J., Computer-Controlled Systems: Theory and Design, PRENTICE HALL, 1996. ISBN: 0133148998
- [3] Marquez, H. J., Nonlinear control systems: analysis and design, volume 1, Wiley-Interscience Hoboken, NJ, 2003. ISBN: 0471427993
- [4] Golnaraghi, F., Kuo, B., Automatic Control Systems, McGraw-Hill Education, New York, 10th edition. edition, 2017. ISSN: 9781259643835
- [5] Sun, X., Xu, J., Fu, J., „The effect and design of time delay in feedback control for a nonlinear isolation system”, *Mechanical Systems and Signal Processing*, 87, pp. 206–217, 2017. doi:<https://doi.org/10.1016/j.ymssp.2016.10.022>
- [6] Meng, H., Sun, X., Xu, J., Wang, F., „The generalization of equal-peak method for delay-coupled nonlinear system”, *Physica D: Nonlinear Phenomena*, 403, p. 132340, 2020. doi:<https://doi.org/10.1016/j.physd.2020.132340>
- [7] Butcher, Numerical Methods for Ordinary Differential Equations, John Wiley & Sons, 2003. ISBN: 0471967580
- [8] Holland, J., Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence, MIT Press, Cambridge, Mass, 1992. ISBN: 9780262082136
- [9] Zadeh, L. A., „Outline of a new approach to the analysis of complex systems and decision processes”, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(1), pp. 28–44, 1973. doi:[10.1109/TSMC.1973.5408575](https://doi.org/10.1109/TSMC.1973.5408575)
- [10] Nielsen, R., Neurocomputing, Addison-Wesley Pub. Co, Reading, Massachusetts, 1990. ISBN: 0201093553

- [11] Zurada, J., Introduction to artificial neural systems, West, St. Paul, 1992. ISBN: 0314933913
- [12] Kurzweil, R., The singularity is near : when humans transcend biology, Viking, New York, 2005. ISBN: 9780143037880
- [13] Fu, J., Zheng, H., Mei, T., „Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition”, in: „2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)”, pp. 4476–4484, 2017. doi:[10.1109/CVPR.2017.476](https://doi.org/10.1109/CVPR.2017.476)
- [14] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., Hassabis, D., „Mastering the game of go without human knowledge”, *Nature*, 550(7676), pp. 354–359, 2017. doi:[10.1038/nature24270](https://doi.org/10.1038/nature24270)
- [15] He, P., Jagannathan, S., „Reinforcement learning neural-network-based controller for nonlinear discrete-time systems with input constraints”, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2), pp. 425–436, 2007. doi:[10.1109/TSMCB.2006.883869](https://doi.org/10.1109/TSMCB.2006.883869)
- [16] Passino, K., Fuzzy control, Addison-Wesley, Menlo Park, Calif, 1998. ISBN: 020118074X
- [17] Nawa, N., Furuhashi, T., „Fuzzy system parameters discovery by bacterial evolutionary algorithm”, *IEEE Transactions on Fuzzy Systems*, 7(5), pp. 608–616, 1999. doi:[10.1109/91.797983](https://doi.org/10.1109/91.797983)
- [18] Thomas, C. M., Nielsen, K. M., „Mechanisms of, and barriers to, horizontal gene transfer between bacteria”, *Nature Reviews Microbiology*, 3(9), pp. 711–721, 2005. doi:[10.1038/nrmicro1234](https://doi.org/10.1038/nrmicro1234)
- [19] Balázs, K., Botzheim, J., Kóczy, L. T., Comparative Investigation of Various Evolutionary and Memetic Algorithms, pp. 129–140, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. doi:[10.1007/978-3-642-15220-7_11](https://doi.org/10.1007/978-3-642-15220-7_11), iSBN: 978-3-642-15220-7
- [20] Billard, L., Diday, E., „Symbolic regression analysis”, in: Jajuga, K., Sokołowski, A., Bock, H.-H., eds., „Classification, Clustering, and Data Analysis”, pp. 281–288, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. ISBN: 978-3-642-56181-8
- [21] Botzheim, J., Cabrita, C., Kóczy, L. T., Ruano, A. E., „Genetic and bacterial programming for b-spline neural networks design”, *J. Adv. Comput. Intell. Intell. Informatics*, 11, pp. 220–231, 2007. doi:[10.20965/jaciii.2007.p0220](https://doi.org/10.20965/jaciii.2007.p0220)

- [22] Yu, X., Gen, M., Introduction to Evolutionary Algorithms, Springer London, 2010. doi:[10.1007/978-1-84996-129-5](https://doi.org/10.1007/978-1-84996-129-5)
- [23] Weiss, M., Data structures and algorithm analysis in C, Addison-Wesley, Menlo Park, Calif, 1997. ISBN: 0201498405
- [24] Coddington, A., Levinson, N., Theory of Ordinary Differential Equations, International series in pure and applied mathematics, McGraw-Hill Companies, 1955. ISBN: 9780070115422
- [25] Ljung, L., System Identification: Theory for the User, Prentice Hall information and system sciences series, Prentice Hall PTR, 1999. ISBN: 9780136566953
- [26] Bolognani, S., Bolognani, S., Peretti, L., Zigliotto, M., „Design and implementation of model predictive control for electrical motor drives”, *IEEE Transactions on Industrial Electronics*, 56(6), pp. 1925–1936, 2009. doi:[10.1109/TIE.2008.2007547](https://doi.org/10.1109/TIE.2008.2007547)