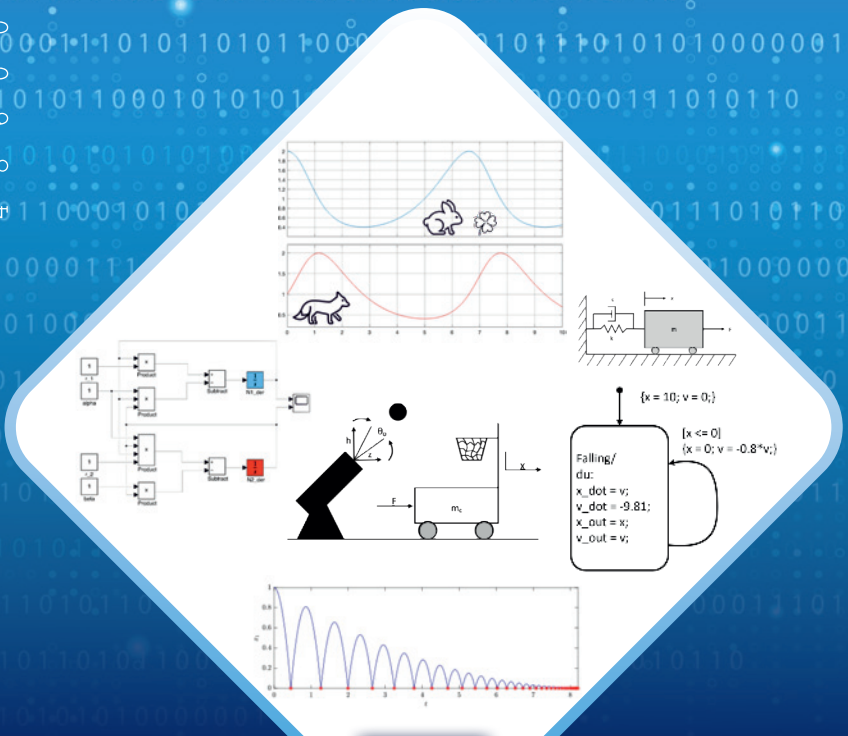


Kmeť Tibor
Csóka Márk
Annuš Norbert
Paksi Dávid

Folyamatmodellezés és számítógépes szimuláció a MATLAB-ban

- 2024 -

ISBN 978-80-8122-503-1

UNIVERZITA J. SELYEHO – SELYE JÁNOS EGYETEM
Fakulta Ekonómie a Informatiky – Gazdaságtudományi és Informatikai Kar

Kmeť Tibor – Annuš Norbert – Csóka Márk – Paksi Dávid

Folyamatmodellezés és számítógépes szimuláció a MATLAB-ban

– 2024 –

Szerzők:
prof. RNDr. Kmeť Tibor, CSc.
Mgr. Annuš Norbert, PhD.
PaedDr. Csóka Márk, PhD.
Mgr. Paksi Dávid, PhD.

Lektorálta:
doc. RNDr. Bukor József, PhD.
Dr. habil. Kiss Attila Elemér, CSc.

Nyelvi lektorálást végezte:
Mgr. Bugár S. Veronika, PhD.

Valamennyi bemutatott modell forráskódja megtalálható az alábbi linken:
https://github.com/JSelyeUniversity/folyamatmodellezes

TARTALOMJEGYZÉK

BEVEZETÉS

A modellezés és szimuláció (MS) egy olyan terület, amelyet az élet számos területén használnak. Segítséget nyújt a tervezés, gyártás, oktatás területén, de sok egyéb szektorban is.

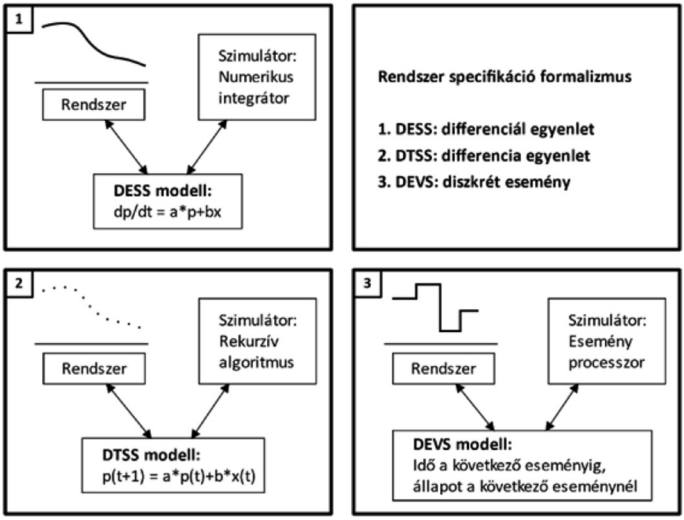
Mindenekelőtt érdemes tisztázni, hogy mit is értünk modell és szimuláció alatt, milyen fajtáit ismerjük, és miért van fontos szerepük a mindennapi életben. A modellezés során a valóságból kiemeljük a számunkra adott szituációban fontos, ismert vagy feltételezett elemeket. Egy populációs modell esetében ilyen paraméter lehet a populáció pillanatnyi nagysága, a születés és elhalálozás gyakorisága és az idő intervallum, amelyen vizsgálni szeretnénk a változást. Fontos, hogy az elkészített modellünket tesztelésnek vessük alá. Amennyiben az összeállított modell az elvártak szerint viselkedik, akkor azt az adott jelenség vizsgálatára alkalmas eszköznek tekinthetjük. Abban az esetben, ha eltéréseket észlelünk, akkor további fejlesztéseket kell végrehajtanunk a modellen, optimalizálnunk kell azt. Megeshet, hogy ezek a módosítások nem segítik az előrehaladást, ekkor a modellt el kell vetni és egy új szemszögből kell megközelíteni annak felépítését. A tesztelések során nagy segítséget nyújtanak a szimulációk. A szimuláció során egy rendszer viselkedését próbáljuk egy másikéval imitálni. Ez egy olyan vizsgálat, amikor egy folyamat fizikai vagy számítógépes modelljén tanulmányozzák a rendszer várható, illetve valódi viselkedését. Egy szimulációt több okból kifolyólag is elvégezhetünk a valós kísérlet helyett, például ha a valós kísérlet végbemenetelének időtartama túl gyors vagy éppen túl lassú lenne. Ezen felül szimuláció kivitelezése sok esetben olcsóbb, mint elvégezni a költséges kísérletet. Szempont lehet az is, ha a kísérletek túl veszélyesek, túl bonyolultak vagy, ha nem állnak rendelkezésre a kivitelezéséhez szükséges eszközök.

Mára már számos modellezésre alkalmas rendszerről beszélhetünk. Adottak olyan modellező szoftverek, melyek egy-egy konkrét terület problémáira fókuszálnak és nyújtanak – modellek és szimulációk segítségével – megoldásokat. Az egyik legalkalmasabb programozási környezet a MATLAB. A benne található eszköztárak közül a Simulink, SimEvents és Stateflow könyvtárak emelhetők ki, amelyek a bennük található blokkok segítségével teszik lehetővé modelleken képzett szimuláció megvalósítását. A modellezni kívánt rendszereket több szempont szerint csoportosíthatjuk. Ezen könyv keretein belül viszont, az időbeli változás szemlélete szerinti kategóriákra helyezzük a hangsúlyt. Ezen csoportok szerint beszélhetünk:

- **Folytonos rendszerekről** (**DESS** - *Differential Equation System Specification*)
- **Diszkrét rendszerekről** (**DTSS** - *Discrete Time System Specification*)
- **Diszkrét eseményű rendszerekről** (**DEVS** - *Discrete Event System Specification*)
- **Hibrid rendszerekről** (**DEV&DESS** - *Discrete Event and Differential Equation System Specification*)

Ezen könyv alapjául a **Bevezetés a modellezés és szimulációba, és a Modellezés és szimuláció elmélete és eszközei** előadásai szolgálnak.

Valamennyi bemutatott modell forráskódja megtalálható az alábbi linken: https://github.com/JSelyeUniversity/folyamatmodellezes



1. Ábra – DESS, DTSS és DEVS modelleket leíró lehetőségek és grafikus ábrázolásainak mintája (Zeigler, Muzy, & Kofman, 2019)

Az első fejezet a matematikai modellalkotás módszereinek bemutatására összpontosít. Ezt követően általánosan ismerteti az egyes rendszertípusokat. A második fejezet már konkrét rendszertípusokkal foglalkozik. Bemutatja a folytonos rendszereket, azon belül is különbséget tesz közönséges differenciálegyenletek és parciális differenciálegyenletek között. Előbbi esetében a rendszer mű-

ködését javarészt populációs- és kocsimodellek segítségével támasztjuk alá, míg utóbbiaknál megnézzük ezen rendszerek numerikus megoldásait. A folytonos rendszereket követve a diszkrét rendszerek modellezésére fektetjük a hangsúlyt. A harmadik fejezet tehát, a diszkrét idejű rendszerek működését foglalja össze, amely magába foglalja a diszkrét idejű térbeli rendszereket is. Ezeket a gyakorlatban matematikai számsorokon, populációs modelleken, valamint egy- és többdimenziós sejtautomatákkal szemléltetjük. A fejezetben helyet kap továbbá Wolfram automatája és a Conway-féle életjáték-modell. A negyedik fejezet a diszkrét eseményű rendszereket mutatja be, tömegkiszolgáló vagy másnéven sorbanállási rendszerek, valamint Markov-láncok segítségével. Az ötödik fejezet a hibrid rendszerekre helyezi a hangsúlyt, melyek folytonos és diszkrét dinamikus viselkedést is produkálnak. Könyvünk hatodik fejezete a MATLAB-ban található modellezési eszköztárakat mutatja be, konkrétan a Simulink-et, a SimEvents-et, valamint a Stateflow-t.

1. MATEMATIKAI MODELLEK ALKOTÁSÁRA SZOLGÁLÓ MÓDSZEREK

Matematikai modell alatt magát a rendszert, és a benne lejátszódó folyamatot leíró egyenleteket értjük. Az ilyen típusú modellek alkotása során figyelembe kell, vegyük a kezdeti és peremfeltételeket, valamint a kapcsolódó adatrendszer felállítását. A modellalkotáshoz sorolhatjuk magát a megoldó algoritmust is, amely meghatározza a modell pontosságát (Giordano, Weir, & Fox, 2003).

Egy adott rendszer modellezése során, a matematikai modellek alkotásakor az alábbi elméleti módszerek adottak (Pokorádi, 2013):

Fehérdoboz-eljárás (White-box): A modell felállítása ezen eljárás esetében fizikai törvényszerűségeken és megfontolásokon alapszik, tehát már a rendszerről kapott előzetes információk alapján történik a matematikai modell előállítása. Itt valós, fizikai paraméterekről beszélhetünk. A módszer hátránya, hogy bonyolult. Felhasználási szempontból elmondható, hogy olyan esetekben alkalmazzuk, amikor részletes tervezésre és pontosságra van szükség.

Feketedoboz-eljárás (Black-box): Tipikusan a kísérleteken alapuló eljárás. Az előző módszerhez képest, itt nem beszélhetünk konkrét fizikai paraméterekről. A kísérlet alatt érthetjük a vizsgálójelre adott rendszerválaszokat és azok elemzését. A white-box-hoz képest ezeknek a modelleknek az előnyük, az egyszerűségükben rejlik. Hátrányuk a valós fizikai törvényszerűségek alkalmazásának hiánya. Mivel ezeket a modelleket a vizsgált rendszer bemeneti és kimeneti paraméterei alapján építjük fel, ezért szakirodalmakban Input/Output, vagy röviden I/O rendszereknek nevezik őket.

Szürkedoboz-eljárás (Grey-box): Az előző két módszer kombinációja, mondhatni egy vegyes eljárás. A mérnöki munkában leggyakrabban használt metódus, mivel, ha nem is mondható el minden paraméter esetében a törvényszerűségen alapuló pontosság, vannak szegmensek melyek esetében valós fizikai paraméterekről beszélhetünk, s vannak, amelyek a black-box eljárásra vannak kényszerülve. Ezen eljárás során az alábbi két alesetet célszerű megvizsgálni (Ljung, 2001):

Fizikai modellezés: Fizikai alapokon felépíthető egy modellstruktúra, amelynek bizonyos számú paraméterét az adatokból kell megbecsülni. Ez lehet például egy adott rendű és szerkezetű állapottérmodell.

Félig fizikai modellezés: A fizikai betekintést a mért adatjelek bizonyos nemlineáris kombinációinak vizsgálatára használják. Ezeket az új jeleket ezután black-box jellegű modellstruktúráknak vetik alá.

A rendszerelmélet alapján beszélhetünk úgynevezett beágyazásokról, melyek szerint alosztályként tekinthetünk az egyik rendszerre a másikhoz viszonyítva. Példaként a *DTSS* a *DEVS* „alosztálya". Azonban a szó szoros értelmében nem igaz, hogy bármelyik diszkrét idejű rendszer egyben diszkrét eseményrendszer is (időbázisaik például különböznek). Lehetőségünk van viszont, olyan szimulációs fogalmak megadására és olyan esetek modellezésére, melyek lehetővé teszik, hogy az egyik rendszer a másik lényegi munkáját elvégezze. Egy formalizmus beágyazható egy másikba, ha az első bármelyik rendszere szimulálható a második valamelyik rendszerével (Zeigler, Muzy, & Kofman, 2019). Visszautalva az előző példánkra, egy *DTSS* szimulálható egy *DEVS*-szel, ha az időbeli előrehaladást állandóra korlátozzuk.

Összegezve tehát, ebben a fejezetben ismertettük a matematikai modell alkotásának lehetőségeit és az egyes rendszertípusokat, melyek eseteiben kitérünk a különálló működésükre, valamint röviden a beágyazási lehetőségeikre is. Lehetőségünk van viszont arra, hogy az ezen rendszerekre jellemző paramétereket egy modellen belül is használjuk, mint pl. a *DEV&DESS*, melyeket egy későbbi fejezetünkben, hibrid rendszerekként mutatunk be (Barros, 1997).

2. FOLYTONOS RENDSZEREK (DESS)

A DESS (*Differential Equation System Specification*), hagyományos differenciálegyenlet-rendszerek, amelyek folytonos állapotokkal és folytonos idővel rendelkeznek. A folytonos folyamatokat az idő vezérli. Az átmenetek folytonos görbe segítségével vannak leírva. A differenciálegyenletekkel leírt modellekben nem közvetlenül a következő állapotot jelöljük, hanem egy olyan függvény deriváltját használjuk, amely meghatározza az állapotváltozó változásának nagyságát. A differenciálegyenletes rendszerek leírása megadható egy struktúraként, ahol

$$DESS = (X, Y, S, f, \lambda)$$

ahol:
- X – bemenetek halmaza
- Y – kimenetek halmaza
- S – állapotok halmaza
- $f: S \times X \rightarrow S$ – a változás mértékét leíró függvény
- $\lambda: S \rightarrow Y$ (Moore-típus) kimeneteket leíró függvény

2.1. Közönséges differenciálegyenlet (ODE)

A differenciálegyenletek gyakori eszközei a valóvilágbéli folyamatok leírásának, ezért a folytonos modelleket többnyire ezek segítségével lehetséges leírni. Az ilyen modellek vizsgálatával a közönséges (*Ordinary Differential Equation*), illetve parciális differenciálegyenletek elmélete foglalkozik.

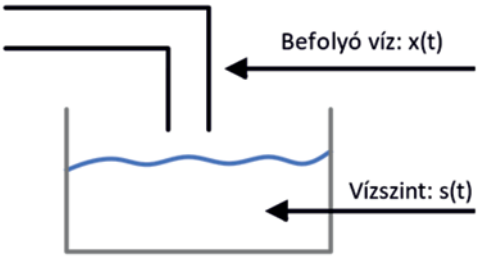
A differenciálegyenlet-modellek esetében egy deriváltfüggvényt használunk az állapotváltozók változási sebességének megadására. Az időtengely bármely adott időpontjában, adott állapot és bemeneti érték esetén, csak az állapot változásának sebességét ismerjük. Ebből az információból kell kiszámítani a jövőbeni, bármely időpontban bekövetkező állapotot.

Ennek a kérdésnek a megvitatásához tekintsük meg a legelemibb folytonos rendszert – az egyszerű integrátort (lásd (2.1)). Az integrátornak egy bemeneti változója *x(t)* és egy kimeneti változója *y(t)* van. Úgy képzelhetjük el, mint egy végtelen kapacitású víztározót. Bármi kerül a tározóba, az felhalmozódik, viszont egy negatív bemeneti érték csökkenti a tároló tartalmát. A tározó kimenete az aktuális tartalma. Amikor ezt egyenlet formájában akarjuk kifejezni,

szükségünk van egy változóra, amely az aktuális tartalmat jelöli. Esetünkben ezt az *s(t)* állapotváltozó reprezentálja. Az *x(t)* aktuális bemenet a tartalom aktuális változásának sebességét jelöli, amit a következő egyenlet fejez ki

$$\frac{ds(t)}{dt} = x(t) \tag{2.1}$$

ahol az *y(t)* kimenet megegyezik az aktuális *y(t) = s(t)* állapottal.



2. Ábra – Egy tároló modell és annak ábrázolása

A folyamatos idejű rendszerek általában több állapotváltozó segítségével fejezhetők ki. A deriváltak az állapotváltozók egy részének, vagy mindegyikének függvényei. Legyenek $s_1, s_2,..., s_n$ az állapotváltozók és $x_1, x_2,..., x_m$ a bemeneti változók, így a folytonos idejű modellt elsőrendű differenciálegyenletek halmaza alkotja

$$\begin{aligned}
\dot{s}_1(t) &= f_1(s_1(t),...,s_n(t), x_1(t),..., x_m(t)) \\
\dot{s}_2(t) &= f_2(s_1(t),...,s_n(t), x_1(t),..., x_m(t)) \\
&\vdots \\
\dot{s}_n(t) &= f_n(s_1(t)..., s_n(t), x_1(t),..., x_m(t))
\end{aligned} \tag{2.2}$$

ahol s_i jelöli ds_i/dt-t. A tömörebb jelölés érdekében az állapotváltozókat csoportosítjuk $s(t) \triangleq [s_1(t),...s_n(t)]^T$ állapotvektorba. Hasonlóképpen, a bemeneti változók írhatók az $x(t) \triangleq [x_1(t),...x_m(t)]$ bemeneti vektorba, így az egyenletek a következőképpen írhatók fel:

$$\begin{aligned}
\dot{s}_1(t) &= f_1(s(t), x(t)) \\
\dot{s}_2(t) &= f_2(s(t), x(t)) \\
&\vdots \\
\dot{s}_n(t) &= f_n(s(t), x(t))
\end{aligned} \tag{2.3}$$

Az egyenletben szereplő f_i függvények egy $f \triangleq [f_1, ...f_n]$ vektorfüggvénybe is csoportosíthatók, így megkapjuk az ODE-k klasszikus állapotegyenlet-ábrázolását:

$$\dot{s}(t) = f(s(t), x(t)) \tag{2.4}$$

A legtöbb folytonos idejű modell egyenlet formájában írható le, amely nem ad explicit értéket a *s(t)* állapotra bizonyos idő elteltével. Így ahhoz, hogy az állapot trajektóriákat megkapjuk, az ODE-t meg kell oldani. A probléma az, hogy az egyenlet megoldása nemcsak nagyon nehéz, hanem egyes esetekben lehetetlen is lehet, mivel csak nagyon kevés ODE-nek van analitikus megoldása ismert függvények és kifejezések formájában. Ez az oka annak, hogy az ODE-ket általában numerikus integrációs algoritmusokkal oldják meg, amelyek közelítő megoldásokat adnak (Mathews & Fink, 1999).

2.1.1. Populációs modellek

Mivel a világ egyre népesebb, a kérdés, hogy „hány embert képes eltartani a Föld, és milyen feltételek mellett?", legalább olyan sürgetővé válik, mint amikor Malthus (1798) a XVIII. század végén felvetette a kérdést az *„An Essay on the Principle of Population"* c. könyvében. A túlnépesedés kérdésének történelmi „megoldásai" két alapfeltevésen nyugodtak: először is, hogy a népességnövekedés állandó, egy főre jutó pozitív arányai mellett a népesség exponenciálisan növekszik, vagyis népességrobbanás figyelhető meg; másodszor, hogy az erőforrások korlátozottsága szükségszerűen szabályozza az ilyen robbanás mértékét. Mindkét feltételezés hasznossága és érvényessége természetesen korlátozott, mivel a környezet, amelyet gyakran „eltartóképességnek" neveznek, nem marad állandó. Az egy főre jutó népességnövekedési ráták nem állandók, hanem a változó környezet függvényei. A népességdinamika hasznos (mind gyakorlati, mind elméleti szempontból) elméletének kialakításában korlátozó tényező az, hogy az elméletalkotók nem képesek olyan modelleket és keretrendszereket biztosítani, amelyekben a környezet plaszticitása érvényesül. Cohen például megjegyzi, hogy a tizennegyedik században a fekete halál, a bubópestis egy formájának ismételt hullámai, valamint a háborúk, a súlyos adók, a felkelések és a rossz, néha rosszindulatú kormányok a becslések szerint az Indiában és Izlandon élő lakosság egyharmadát megölték, és hogy Mezo-Amerika lakossága a tizenhatodik században talán 80-90 százalékkal csökkent. Az ilyen rövid távú, éles csökkenések ellenére azonban, a világ népességének nagysága valójában az őskor óta folyamatosan nőtt, bár nem állandó ütemben.

Az összetett demográfiai folyamatok által felvetett kérdéseket és kihívásokat matematikai modellek segítségével lehet gyakorlati és koncepcionális módon kezelni. A modellek akkor lehetnek különösen értékesek, ha a demográfusokkal, szociológusokkal, közgazdászokkal és egészségügyi szakértőkkel való együttműködés áll a modellépítés középpontjában. Az egyszerű modellek természetüknél fogva nem képesek a fentiekben leírt tényezők közül egyszerre sokakat figyelembe venni. Azonban gyakran hasznos meglátásokkal szolgálnak és segítik az összetett folyamatok megértését. Az egyszerű modellek előrejelzésre való hasznossága korlátozott, és használatuk gyakran félrevezető eredményekhez vezethet a „feketedoboz" felhasználóinak kezében. Az olyan egyszerű népesedési modellek, mint a Malthus-féle (exponenciális) és a Verhulst-féle (logisztikus) modellek természetes kiindulópontot jelentenek a demográfiai folyamatok tanulmányozásában. Fő szerepük itt az, hogy segítsenek megérteni a társadalom- és természettudományok alapvető, idealizált demográfiai jelenségeinek dinamikáját (Brauer & Castillo-Chavez, 2012), (Kmeť, Modellezés és Szimuláció 2020, 2018) (Kmeť, Modellezés és szimuláció elmélete és eszközei, 2021), (Kmeť, Mathematical Modelling and Simulation of Biological Systems, 2005).

2.1.a. Példa - Exponenciális növekedés modellje

Legyen

$$\dot{s} = g(s) \cdot s(t)$$

ahol $g(s)$ jelöli a növekedési rátát, melyet állandónak (konstans) tekintünk és továbbiakban r-rel jelöljük, valamint $s(t)$ jelenti a populáció nagyságát t időben. A növekedési ráta kiszámítható a születések és a halálozások különbségéből.

$$r = g(s) = b - d$$

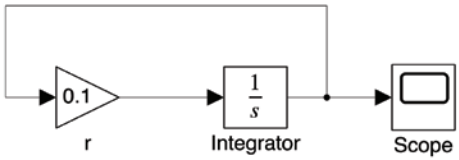
A végleges képlet így

$$\dot{s} = rs(t) \tag{2.5}$$

ahol:

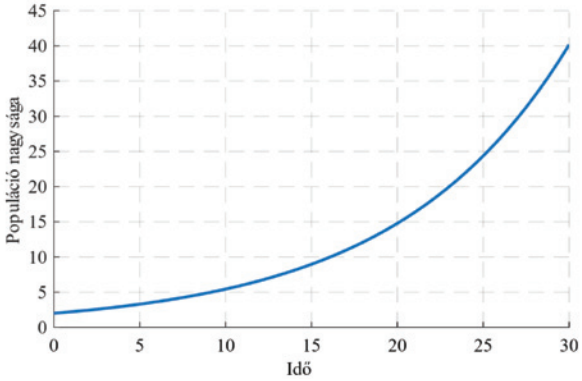
- $s(t)$ – Populáció nagysága
- r – Növekedési ráta
    - $r = b - d$
    - b – Születések száma
    - d – Halálozások száma

Simulink modell:



3. Ábra – Exponenciális modell megvalósítása Simulink-ben

Szimuláció kimenete *r = 0.1* és *s(0) = 2* esetén:



4. Ábra – Exponenciális modell szimulációja

2.1.b. Példa - Logisztikus növekedés modellje

Az exponenciális növekedés modelljének megvannak a felhasználási területei, azonban számos alkalommal nem reális adatokat közöl. Ennek a problémának a feloldására vezetjük be az *R(s)* regulációs faktort.

$$R(s) = 1 - \frac{s}{K} \tag{2.6}$$

R(s) tulajdonságai:
- *R(s) = 1, ha s = 0*
- *R(s) > 0, ha s < K*
- *R(s) = 0, ha s = K*
- *R(s) < 0, ha s > K*

ahol *K* jelöli a kapacitást.

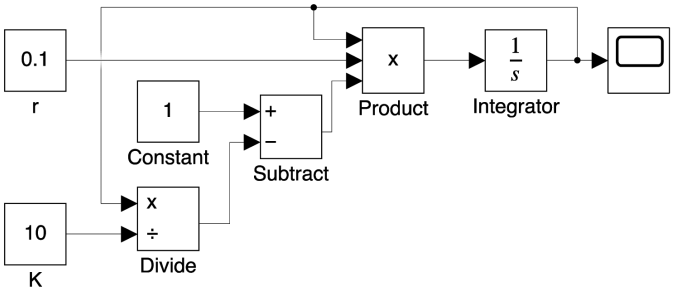
Az exponenciális egyenlet kibővítve a regulációs faktorral vagy eltartóképességgel:

$$\dot{s} = r \cdot s \cdot R(s)$$

A behelyettesítés után a már jól ismert formát kapjuk:

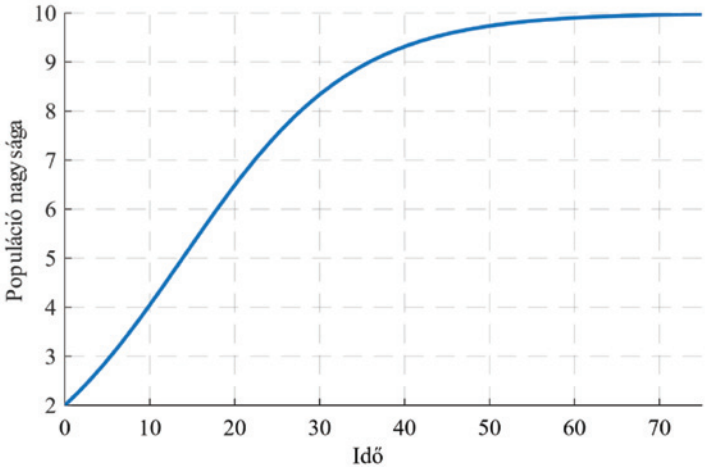
$$\dot{s} = rs\left(1 - \frac{s}{K}\right) \tag{2.7}$$

Simulink modell:



5. Ábra – Logisztikus növekedés modell megvalósítása Simulink-ben

Szimuláció kimenete $r = 0.1$, $K = 10$ és $s(0) = 2$ esetén:



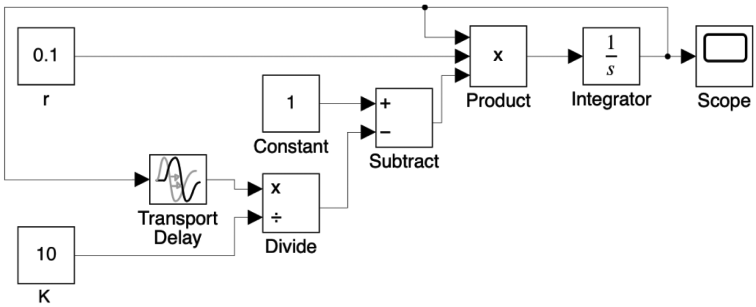
6. Ábra – Logisztikus növekedés modell szimulációja

2.1.c. Példa - Logisztikus növekedés modellje τ időeltolással

Eddig a folytonos populációs modellek tanulmányozása során azt feltételeztük, hogy $\dot{s}(t)$, a populáció méretének növekedési üteme egy adott t időpontban csak az $s(t)$-től függ. Vannak azonban olyan helyzetek, amikor a növekedési ráta nem reagál azonnal a populáció méretének változására. Ha feltételezzük, hogy az $\dot{s}(t)/s(t)$ kapacitásonkénti növekedési ráta az $s(t-\tau)$ függvénye, ahogyan ez például egy olyan populáció modellezésénél lehetséges, amelynek táplálékellátásához τ időre van szükség, egy olyan modellhez jutunk, amely a következő formájú:

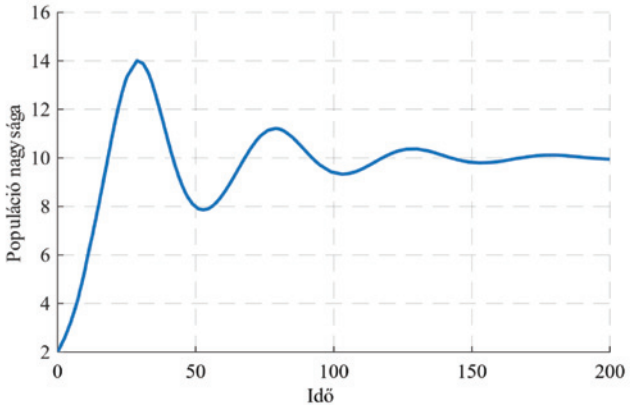
$$m = 2,\ M = 5 \tag{2.8}$$

Simulink modell:



7. Ábra – Késletetett logisztikus növekedés modell megvalósítása Simulink-ben

A szimuláció kimenete $m = 2$, $M = 5$ esetén:



8. Ábra – Késletetett logisztikus növekedés modell szimulációja

2.1.d. Példa - Predáció modell (Lotka-Volterra)

Az 1920-as években Vito Volterrát megkérdezték, hogy lehetséges-e magyarázatot találni az Adriai-tenger halállományában megfigyelt ingadozásokra, amelyek a halászokat nagy aggodalommal töltötték el, amikor a halállomány alacsony volt. Volterra (1926) megalkotta azt a modellt, amely Lotka-Volterra-modell néven vált ismertté (mivel A. J. Lotka (1925) körülbelül ugyanebben az időben egy másik kontextusban hasonló modellt alkotott), és amely azon a feltételezésen alapult, hogy a halak és a cápák ragadozó-zsákmány viszonyban állnak egymással.

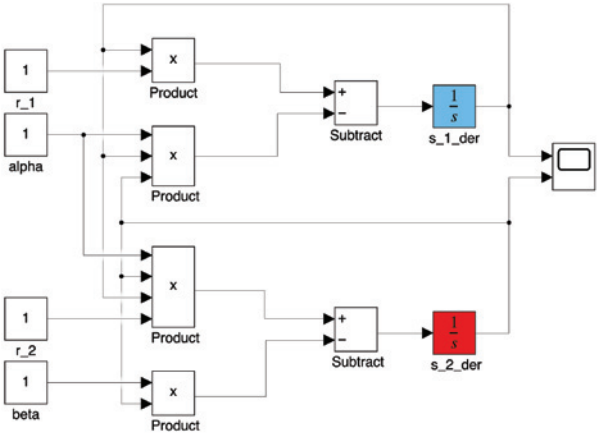
Feltételezzük, hogy a zsákmány növekedése a ragadozó nélkül exponenciális. A ragadozó megjelenésével a zsákmány populációja lecsökken, majd ezt követően a kevés élelem miatt a ragadozóé is. Mindez folytatódik periodikusan. Lehetséges kimenetelek lehetnek még az egyes fajok, vagy csak a ragadozó faj kihalása. Az ezt leíró modell:

$$m = 5, M = 15 \tag{2.9}$$

ahol:

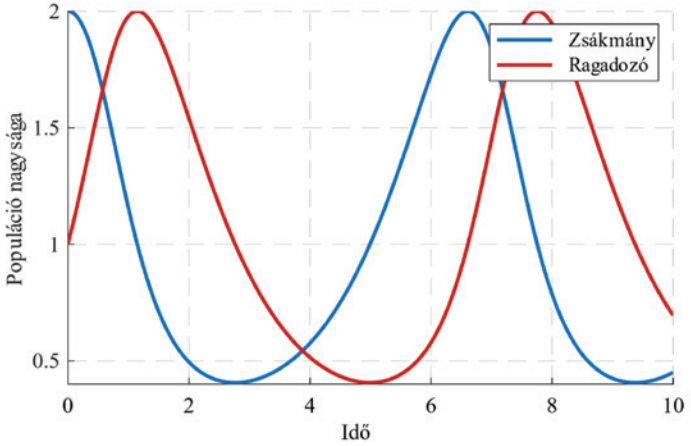
- s_i – i-edik populáció egyedszáma
- r_i – i-edik populáció növekedési rátája ($i = 2$ esetén ez a zsákmány hasznosításának a mértékét jelzi)
- α – A ragadozó populáció zsákmányra gyakorolt hatása
- β – Ragadozók halálozási rátája

Simulink modell:



9. Ábra – Predáció modell megvalósítása Simulink-ben

Szimuláció kimenete $r_1 = 1$, $r_2 = 1$, $\alpha = 1$, $\beta = 1$, $s_1 = 1$, $s_2 = 2$ esetén:



10. Ábra – Predáció modell szimulációja

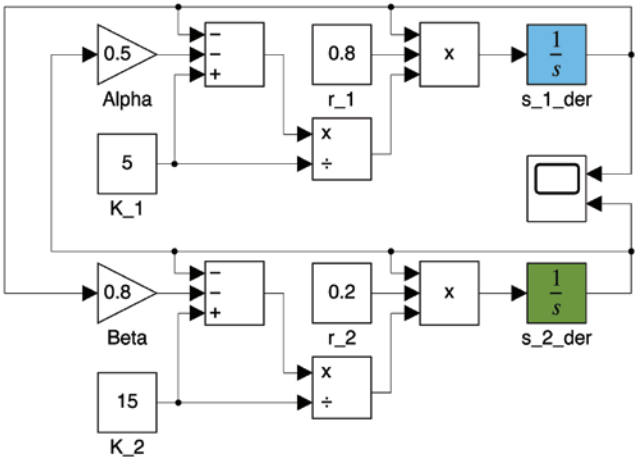
2.1.e. Példa - Kompetíció modell

A fajok versengésére vonatkozó klasszikus kísérleti eredmény a G. F. Gause (1934) által megfogalmazott, kompetitív kizárás elve, amely szerint két, ugyanazon forrásért versengő faj nem létezhet egymás mellett. A kísérleti bizonyítékok némileg kétértelműek, és jelentős kétségek merülnek fel az elv egyetemességét illetően. Ebben a modellben két egymásra közvetlenül nem ártalmas faj viselkedését vizsgáljuk. Mindkét populáció ugyanazon erőforrásért vetekszik, mivel ez a túlélésük záloga. Közvetve a két faj hat egymásra, ugyanis a populációjuk nagyságával az erőforrásigényük is nő. Ugyanezen erőforrásért harcol mindkét faj. Ennek egy lehetséges felírása a logisztikus növekedés modelljét alapul véve az alábbi:

$$\begin{aligned} \dot{s}_1 &= r_1 s_1 \frac{K_1 - s_1 - \alpha s_2}{K_1} \\ \dot{s}_2 &= r_2 s_2 \frac{K_2 - s_2 - \beta s_1}{K_2} \end{aligned} \tag{2.10}$$

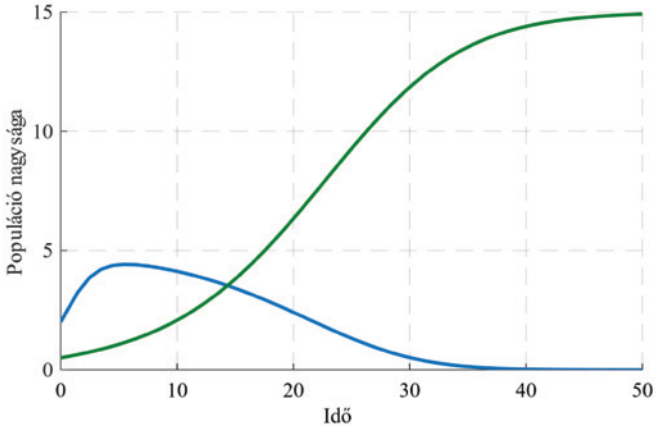
- s_i – i-edik populáció egyedszáma
- r_i – i-edik populáció növekedési rátája
- α – A második populáció elsőre gyakorolt hatása
- β – Az első populáció másodikra gyakorolt hatása
- K_i – i-edik populáció kapacitása

Simulink modell:



11. Ábra – Kompetíció modell megvalósítása Simulink-ben

Szimuláció kimenete $r_1 = 0.8$, $K_1 = 5$, $r_2 = 0.2$, $K_2 = 15$, $\alpha = 0.5$, $\beta = 0.8$, $s_1 = 2$, $s_2 = 0.5$ esetén:



12. Ábra – Kompetíció modell szimulációja

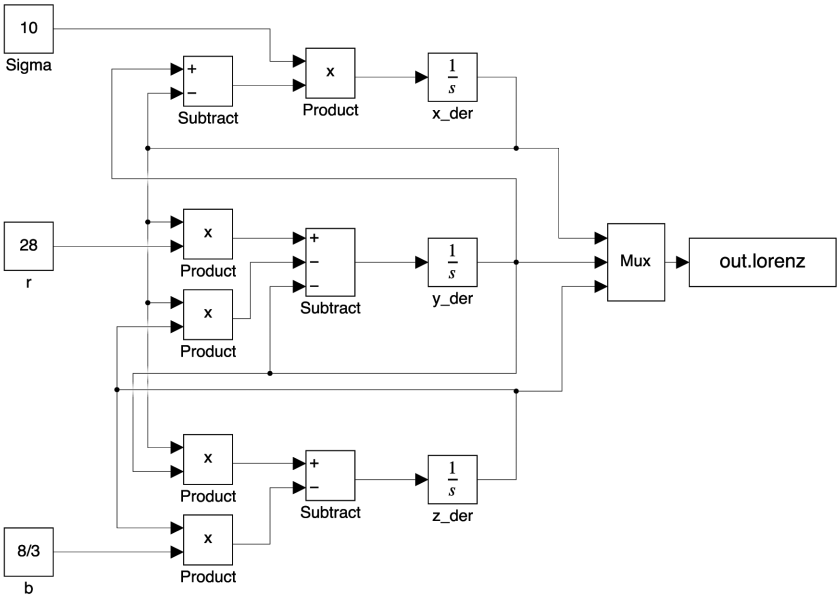
2.1.2. Lorenz attraktor

A modell, folyadékban lebegő részecskék helyzetét ragadja meg. A részecskét a föld alulról melegíti, így az felfelé száll, majd fent lehűl és leereszkedik. A modell egy kétdimenziós folyadék/gáz-cella fizikai folyamatainak leegyszerűsítése, amiben csak 3 változó mennyiség van. Ezek közül az egyik a hőáramlás általi körülfordulás intenzitása (x), a másik kettő pedig a vízszintes és függőleges hőmérsékletváltozás (y és z). A modellt tehát egy háromdimenziós, nemlineáris, determinisztikus, közönséges differenciálegyenlet-rendszer adja meg (Gyenge, 2010):

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= rx - y - xz \\ \dot{z} &= xy - bz \end{aligned} \tag{2.11}$$

Az egyenletrendszer három paramétere a σ Prandtl szám, az r Rayleigh szám és a b paraméter, ami a rendszer méretét szabályozza.

Simulink modell:



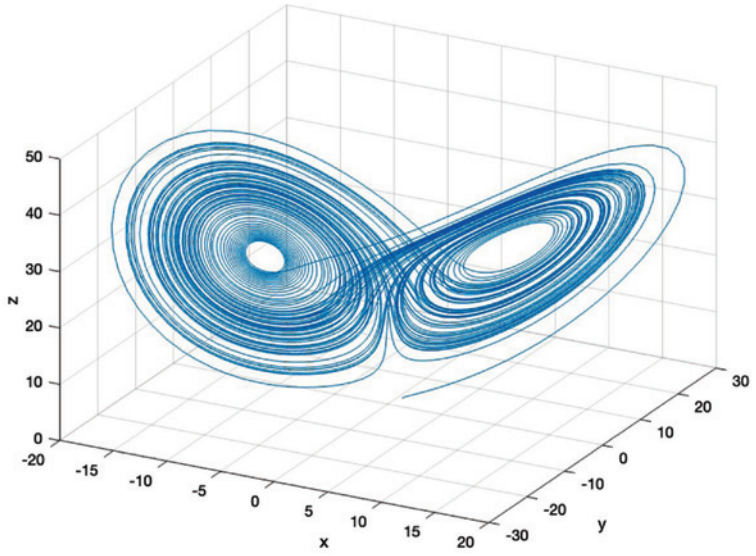
13. Ábra – Lorenz attraktor megvalósítása Simulink-ben

MATLAB kód:

1. Forráskód – Lorenz attraktor szimuláció adatainak megjelenítése

```
close all, clc

x = out.lorenz.data(:,1);
y = out.lorenz.data(:,2);
z = out.lorenz.data(:,3);

plot3(x,y,z)
xlabel x
ylabel y
zlabel z

grid on
```

Szimuláció kimenete:



14. Ábra – Lorenz attraktor szimulációja

2.1.3. Kocsi modellek

A fizikai jelenségek modellezése kiemelkedően nagy figyelmet kapott az idők során. Ezen modellek megléte nélkül nem szállhattunk volna a Holdra, nem állíthatnánk műholdakat Föld körüli pályára, vagy nem lehetne távolságkövető tempomat az autóinkban. A következőkben tekintsünk meg néhány ilyen technológia alapjául szolgáló modellt (B. Dabney & L. Harman, 2004) (Kmeť, Modellezés és szimuláció elmélete és eszközei, 2021).

2.1.f. Példa - Egyszerű kocsi modell

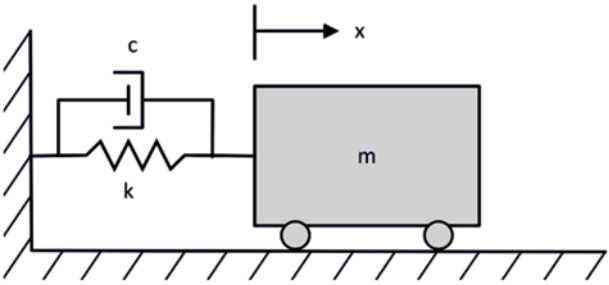
Tegyük fel, hogy a csillapítási együttható $c = 1\ \mathrm{kg/m}$, a rugóállandó $k = 2\ \mathrm{kg/m}$ és a kocsi tömege $m = 5\mathrm{kg}$. A rendszerbe nincsenek bemenetek. Modellezni fogjuk a kocsi mozgását, feltételezve, hogy alapvetően 1 méterre eltér az egyensúlyi állapotától.

Annak érdekében, hogy modellezzük a rendszert, szükséges a mozgás egyenletét felírni. A newtoni megközelítést használva megfigyelhetjük, hogy vízszintes irányban két erő hat a kocsira: rugóerő és a súrlódási erő. A rugóerő *kxa* súrlódási erő pedig $c\dot{x}$.

$$
\begin{aligned}
&x - \text{távolság}\\
&\dot{x} - \text{sebesség}\\
&\ddot{x} - \text{gyorsulás}
\end{aligned}
\tag{2.12}
$$

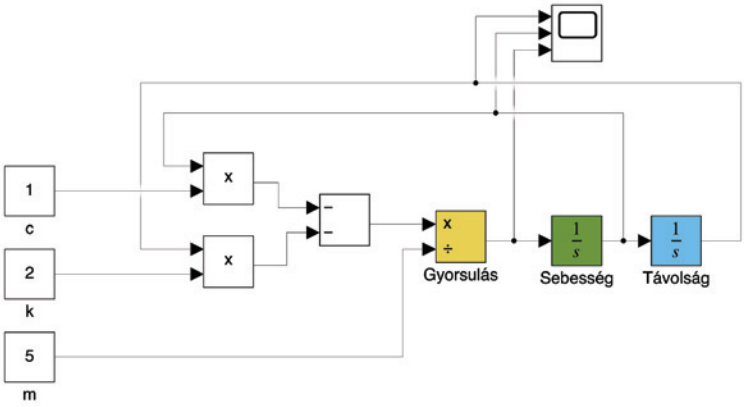
$$
m\ddot{x} = -c\dot{x} - kx
$$

$$
\ddot{x} = \frac{-c\dot{x} - kx}{m}
$$



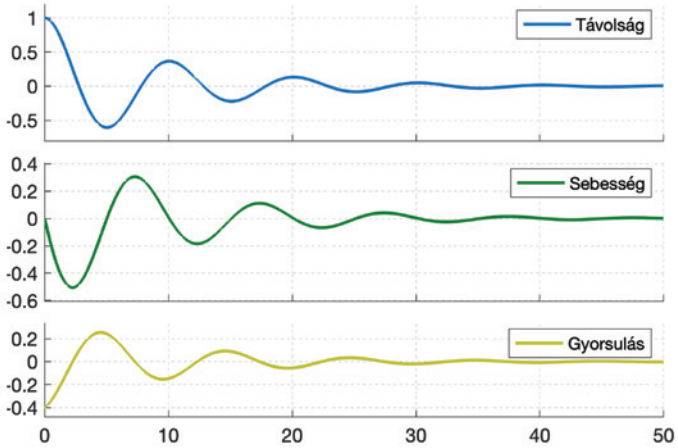
15. Ábra – Egyszerű kocsi modell

Simulink modell:



16. Ábra – Egyszerű kocsi modell megvalósítása Simulink-ben

Szimuláció kimenete $c = 1, k = 2, m = 5, x(0) = 1, \dot{x}\,(0) = 0$ esetén:

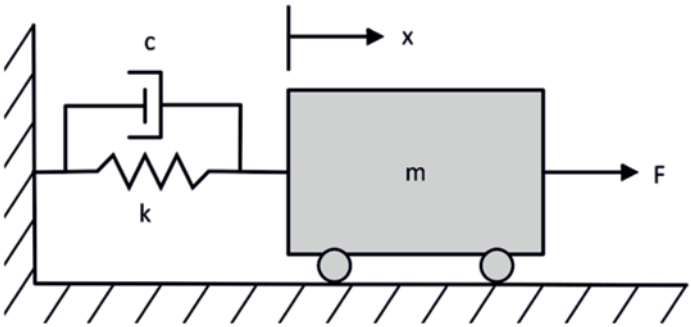


17. Ábra – Kocsi modell szimulációja

2.1.g. Példa - Kocsi modell külső erőhatással

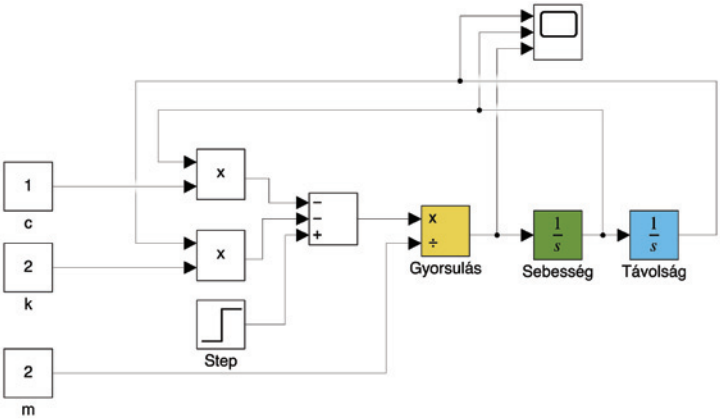
Az alábbi modell megegyezik az előzővel, csak kiegészítjük a kocsira ható vízszintes irányú Ferővel. Tegyük fel, hogy a rendszer alapvetően egyensúlyi állapotban van *(x = 0, ẋ = 0)*, és a kényszerítő funkció egy „Step” bemenet, amely 1 kg.

$$m\ddot{x} = F - c\dot{x} - kx$$
$$\ddot{x} = \frac{F - c\dot{x} - kx}{m} \tag{2.13}$$



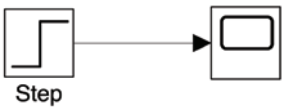
18. Ábra – Kocsi modell külső erőhatással

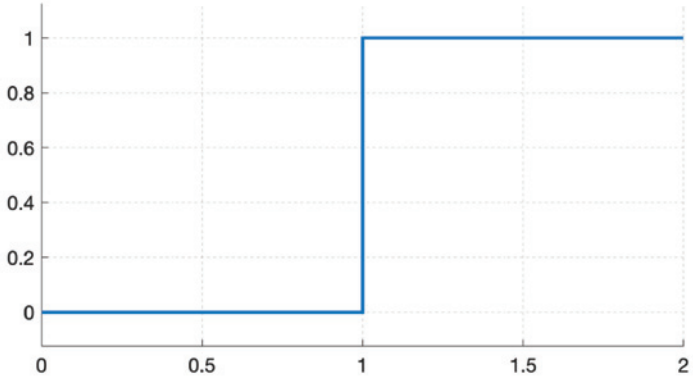
Simulink modell:



19. Ábra – Kocsi modell megvalósítása külső erőhatással Simulink-ben

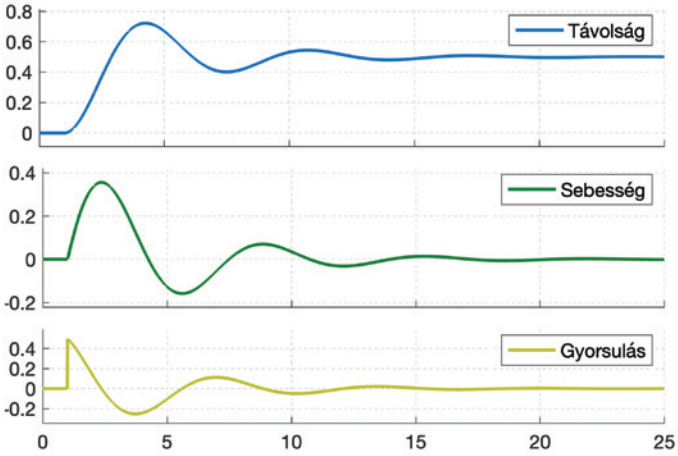
Step működése:





20. Ábra – Step blokk működése

Szimuláció kimenete $c = 1$, $k = 2$, $m = 2$, $x(0) = 0$, $\dot{x}(0) = 0$ esetén:



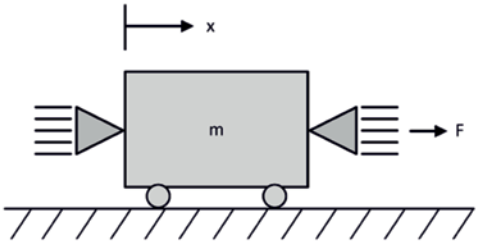
21. Ábra – Kocsi modell külső erőhatással szimulációja

2.1.h. Példa - Kocsi modell rakétával

A következőkben nem rögzítjük a kocsit, egy szabad pályán futhat. Helyezzük le egy tetszőleges helyre. Cél, hogy a kocsit eljuttassuk a 0 pozícióba. Ez akkor fog teljesülni, amennyiben a pozíciója és a sebessége is egyidőben 0-hoz tetszőlegesen közel vannak.

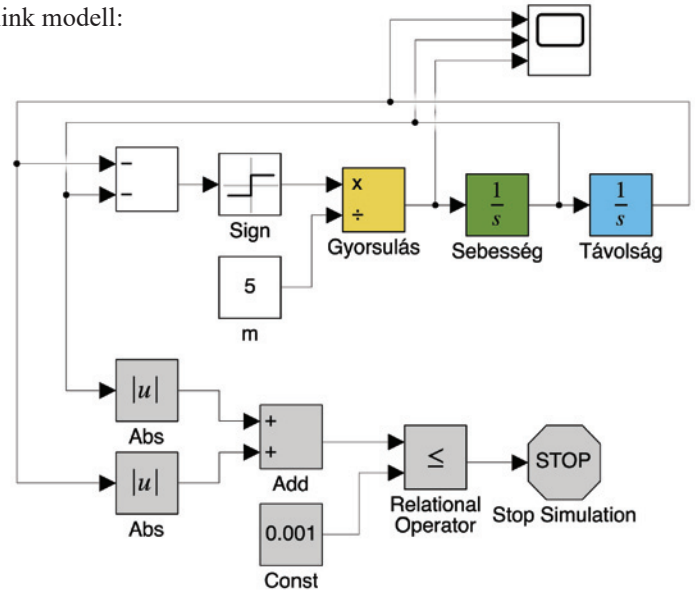
A modell megalkotását az előzőkhez hasonlóan 2 db „integrator” blokk lehelyezésével kezdjük. Az első integrátor blokk bemenete a sebesség lesz. A gyorsulás mozgásegyenletének megoldása: $\ddot{x} = mF$

Adjunk egy „Gain” blokkot hozzá, hogy megszorozzuk a motor erejét $1/m$-el. A „Gain” blokk bemenete F, a motor ereje lesz. A motor ereje $F = 1.0$, ha a gyorsulás és az elmozdulás összege negatív és -1.0, ha az összeg pozitív. Építhetünk egy megfelelő „bang-bang” kontrollert a „Sum” és a „Sign” blokkokat felhasználva. A „Sign” blokk kimenete 1.0, ha a bemenet pozitív és -1.0, ha az input negatív, valamint 0.0 ha a bemenet 0.



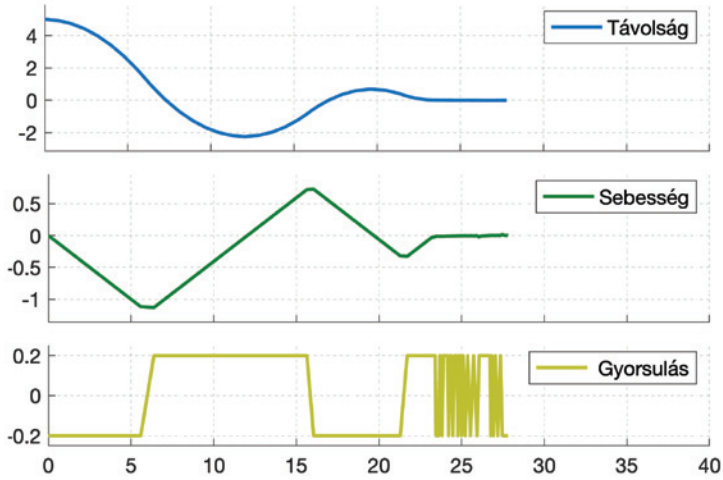
22. Ábra – Kocsi modell rakétával

Simulink modell:



23. Ábra – Rakétás kocsi modell megvalósítása Simulink-ben

Szimuláció kimenete $m = 5, x(0)=5,\ \dot{x}(0)=0$ esetén:



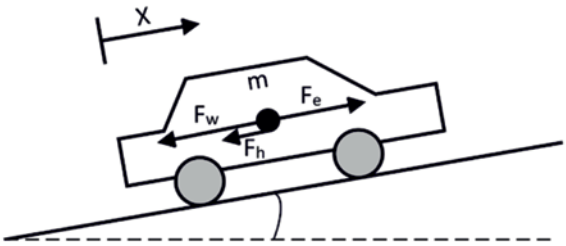
24. Ábra – Rakétás kocsi modell szimulációja

2.1.i. Példa - Tempomat modell

A következőkben legyen a cél egy olyan rendszer megalkotása, mely képes egy autó sebességét bizonyos értékek között tartani.

Az alábbi autó egyenesen halad egy emelkedőn. Három különböző erő hat az autóra: az emelkedőhöz képest vízszintes irányban a motor által produkált húzóerő (vagy fékező erő, ha negatív) (F_e), a közegellenállás (F_w), és a súlyerő tangenciális komponense (F_h). Alkalmazva Newton második törvényét, az autó mozgásegyenlete az alábbi:

$$m\ddot{x} = F_e - F_w - F_h \tag{2.14}$$



25. Ábra – Tempomat modell

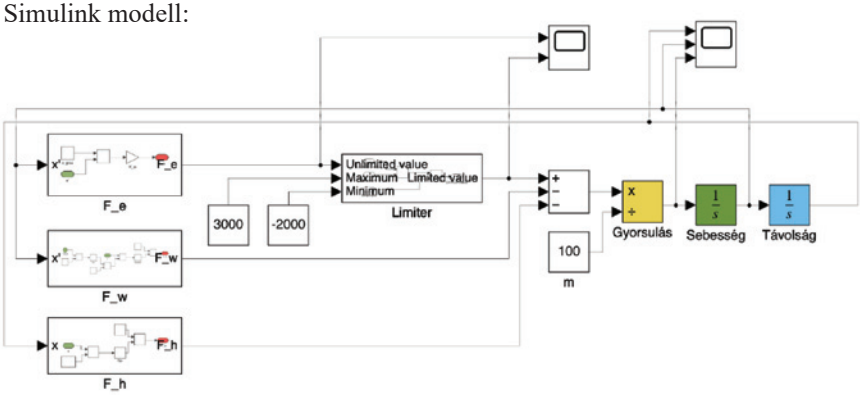
ahol m az autó tömegét jelzi és $\ddot{x}$ pedig a megtett távot. F_e alulról és felülről is korlátozva van. A felső korlát az a maximális erő, melyet a motor, a kerék által a föld irányába tud kifejteni, az alsó pedig a maximális fékezési erő. Feltételezni fogjuk, hogy $-2000\text{N} \leq F_e \leq 1000\text{N}$, ahol N a Newton mértékegységet jelöli és az autó súlya 1000 kg.

$$\ddot{x} = \frac{F_e - F_w - F_h}{m}$$

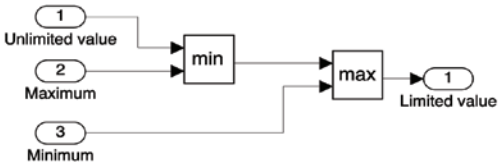
$$F_e = (x_{poz} - \dot{x}) \quad ahol \quad F_e \in [-2000, 3000]$$
$$F_w = 0.01(\dot{x} + 20\sin(0.01t))^2$$
$$F_h = 30\sin(0.0001x) \qquad (2.15)$$

ahol:
- F_e – húzóerő
- F_w – közegellenállás
- F_h – a súlyerő tangenciális komponense
- m – autó tömege

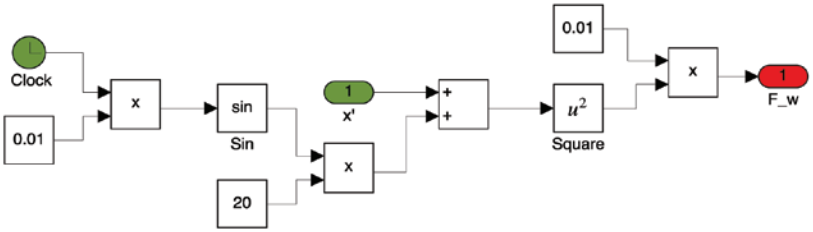
Simulink modell:



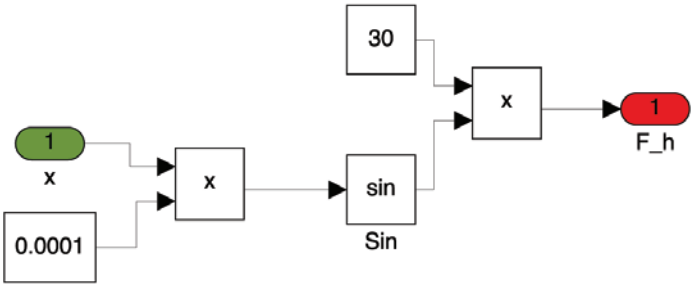
26. Ábra – Tempomat modell megvalósítása Simulink-ben



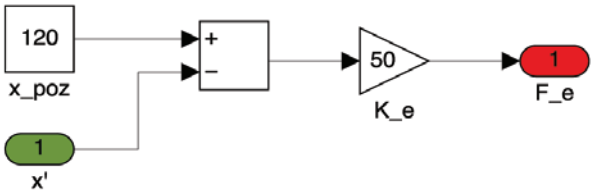
27. Ábra – Értékek limitálása



28. Ábra – Autóra ható aerodinamikai erő

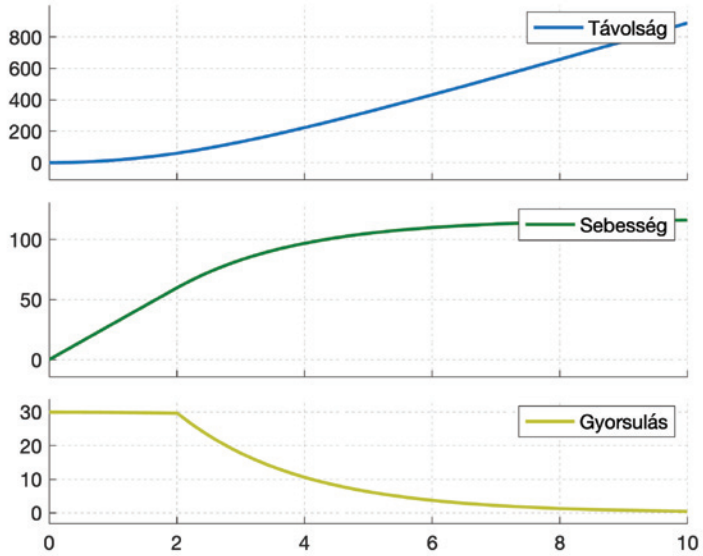


29. Ábra – A gravitáció tangenciális komponensei



30. Ábra – Motor és fékerő

Szimuláció kimenete $m = 100, x(0) = 0, \dot{x}(0) = 0$ esetén:

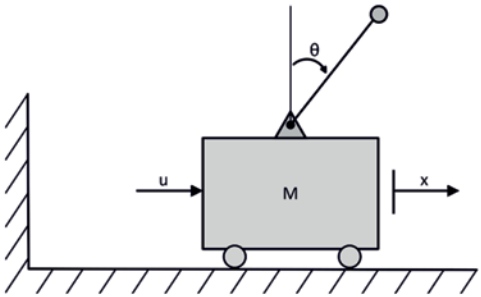


31. Ábra – Tempomat modell szimulációja

2.1.j. Példa - Fordított inga egyensúlyozása kocsin

Vegyük alapul az előzőekben használt rakétás kocsi modellt (2.1.h.) annyi kiegészítéssel, hogy a tetejére felhelyezünk egy ingát, melynek a szára merev. Cél, hogy az ingát egyensúlyozzuk a kocsi tetején.

A rendszer bemenete a vízszintes erő (u), amely hat az M tömegű kocsira. Az inga szabadon forog súrlódás nélkül az oldal síkjában. Az inga hossza l, a tömege pedig (m). Feltételezzük, hogy a súly a végén koncentrálódik. Ennek a rendszernek a kocsi elmozdulására és az inga szögére vonatkozó mozgásegyenlete az alábbi:



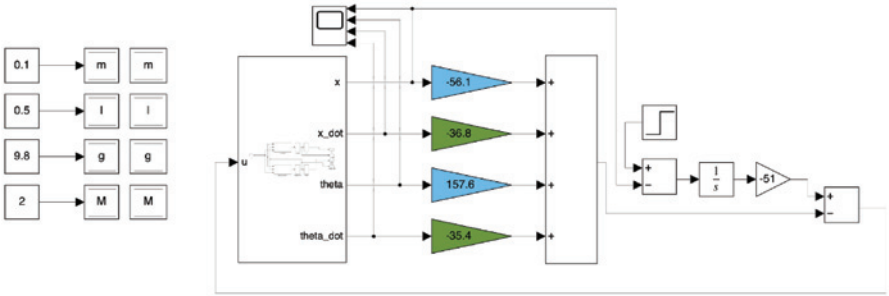
32. Ábra – Fordított inga egyensúlyozása kocsin

$$(M+x)\,\ddot{x} - ml\dot{\theta}^2 - ml\theta + ml\ddot{\theta}cos\theta = u$$
$$m\ddot{x}\,cos\theta + ml\ddot{\theta} = mgsin\theta$$

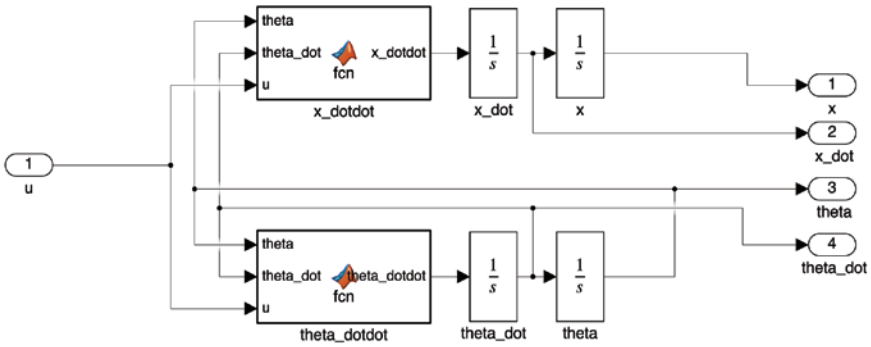
$$\ddot{x} = \frac{ml\dot{\theta}^2\,sin\theta - mgsin\theta cos\theta + u}{M + msin^2\,\theta}$$
$$\ddot{\theta} = -\frac{ml\dot{\theta}^2\,sin\theta cos\theta - mgsin\theta + ucos\theta - Mgsin\theta}{l(M + sin^2\,\theta)} \qquad (2.16)$$
$$u = K_1 \int_0^T (x_{step} - x(t)dt)$$

$$k_1 x + k_2 \dot{x} + k_3 \theta + k_4 \dot{\theta}$$

Simulink modell:



33. Ábra – Kocsin egyensúlyozó fordított inga modell megvalósítása Simulink-ben



34. Ábra – Kocsin egyensúlyozó fordított inga modell alrendszere

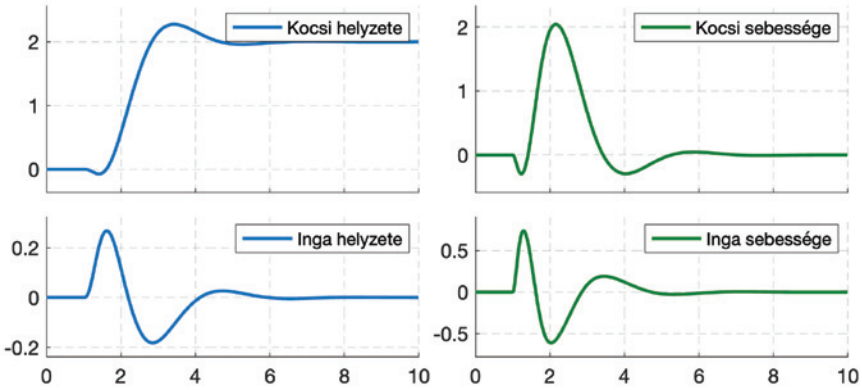
2. Forráskód – x_dotdot függvény

```
function x_dotdot = fcn(theta,theta_dot,u)
global m l g M
x_dotdot = (m*l*theta_dot^2*sin(theta)-
m*g*sin(theta)*cos(theta)+u)/(M+m*sin(theta)^2);
end
```

3. Forráskód – theta_dotdot függvény

```
function theta_dotdot = fcn(theta,theta_dot,u)
global m l g M
theta_dotdot = -(m*l*theta_
dot^2*sin(theta)*cos(theta)-
m*g*sin(theta)+u*cos(theta)-M*g*sin(theta))/
(l*(M+m*sin(theta)^2));
end
```

Szimuláció kimenete $m = 0.1, \mathrm{l} = 0.5, g = 9.8, M = 2, x(0) = 0, \dot{x}(0) = 0, \theta = 0,$ $\dot{\theta}$=0 esetén:

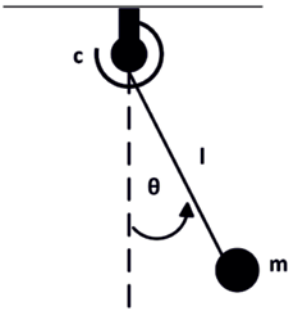


35. Ábra – Fordított inga egyensúlyozása kocsin modell szimulációja

2.1.k. Példa - Csillapított inga

A következőkben egy csillapított ingát modellezünk. Az inga egy fix pontban van felfüggesztve. A rugalmatlan kötél hosszát jelölje *l*. Tegyük fel, hogy az inga tömege pontszerűen a kötél végére összpontosul, melyet *m*-el jelölünk. Ahhoz,

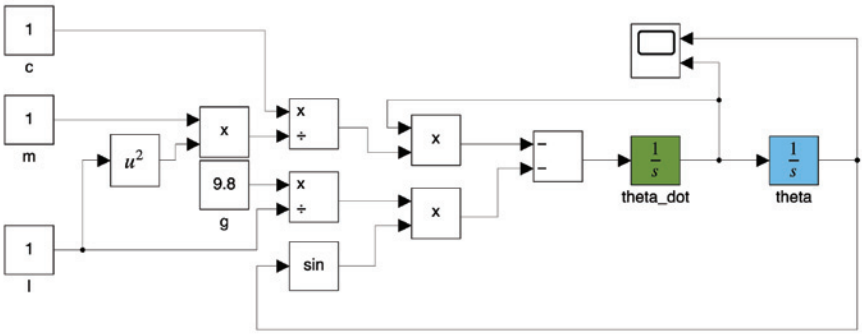
hogy az inga ne a két végállása között mozogjon a végtelenségig, fékezzük az ingát a *c*-vel jelölt csillapítási együtthatóval. Cél, hogy a csillapítást úgy állítsuk be, hogy az inga minél hamarabb megálljon. A modell felépítését a 36. Ábra szemlélteti.



36. Ábra – Csillapított inga

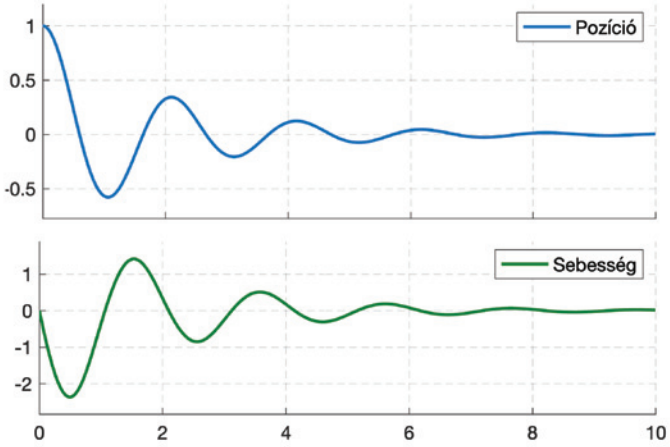
$$\ddot{\theta} = -\frac{c}{ml^2}\,\dot{\theta} - \frac{g}{l}\sin\theta \tag{2.17}$$

Simulink modell:



37. Ábra – Csillapított inga megvalósítása Simulink-ben

Szimuláció kimenete $c = 1, l = 1, m = 1, g = 9.8, \dot{\theta} = 0, \ddot{\theta} = 1$esetén:



38. Ábra – Csillapított inga szimulációja

2.1.l. Példa - Kosárlabda elkapása

A következőkben egy egyenes vonalon haladó kocsira szerelt kosárlabda palánkba szeretnénk beletalálni, egy kosárlabdával. A labdát egy ágyúból lőjük ki. A cél, a megfelelő kilövési erő és szög meghatározása.
A kocsi mozgásegyenlete

$$\ddot{x} = \frac{F}{m_c} \tag{2.18}$$

ahol:

- $F = 20N, \quad x < 5m/s$
- $F = 0, \quad x \geq 5m/s$
- m_c – a kocsi tömege.

A labda kilövési sebessége 15m/s a labdára hat a súlyerő, valamint a közegellenállás. A labda kezdeti magassága 0.5m-rel a kosár felett van. Vízszintes irányban (z koordináta) a labda mozgásegyenlete

$$\ddot{z} = -F_d \cos\theta / m_b \tag{2.19}$$

és a függőleges irányban (h koordináta)

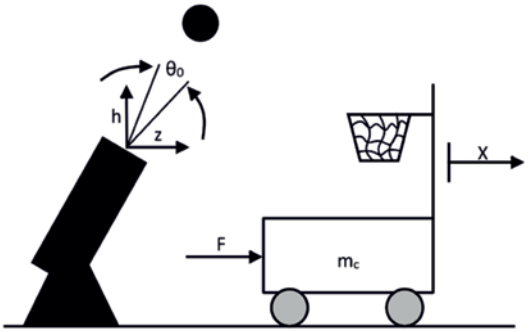
$$\ddot{h} = g - F_d \sin\theta / m_b \tag{2.20}$$

amíg a labda felfelé repül, addig a mozgásegyenlet $m\ddot{h} = mg + F_d\, sinv$, mert a közegellenállás a mozgással ellentétes irányban hat. Amikor a labda már túljutott pályája csúcsán és lefelé zuhan, akkor a mozgásegyenlet $m\ddot{h} = mg - F_d$ sinv, ahol F_d az közegellenállás

$$F_d = \frac{1}{2} C_d A_b\, pv^2 \tag{2.21}$$

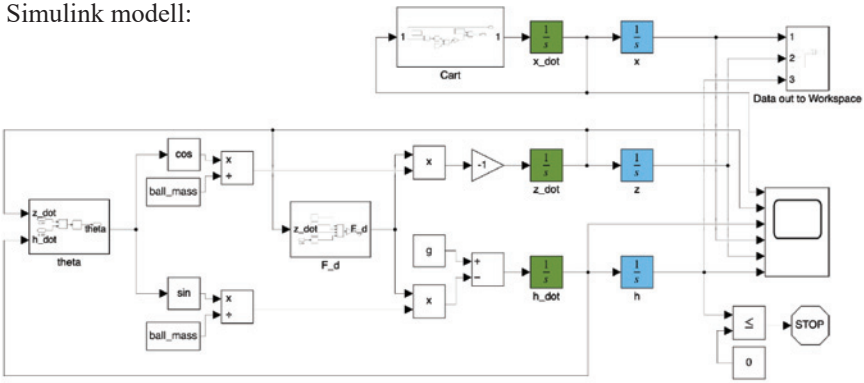
és v a labda sebessége, C_d ellenállási együttható legyen 1.0, A_b a 0.15m sugarú labda keresztmetszetének a területe, és p a levegő sűrűsége (1.224kg/m^3). θ a labda pillanatnyi repülési szöge és v_0 a labda kezdeti sebességének a vízszintes síkkal bezárt szöge.

$$\theta = tan^{-1}\,(\dot{h} / \dot{z}) \tag{2.22}$$

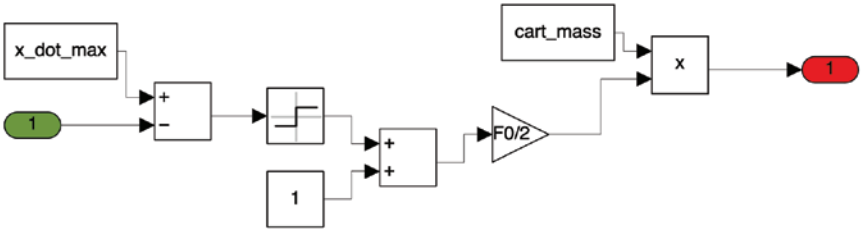


39. Ábra – Kosárlabda elkapása

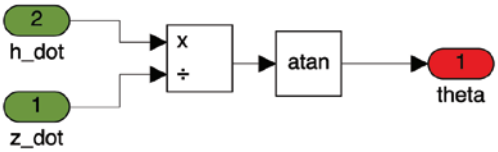
Simulink modell:



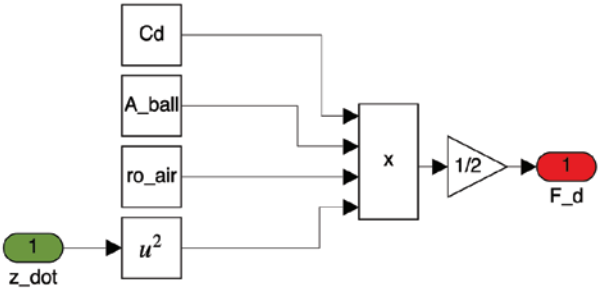
40. Ábra – Kosárlabda elkapás modell megvalósítása Simulink-ben



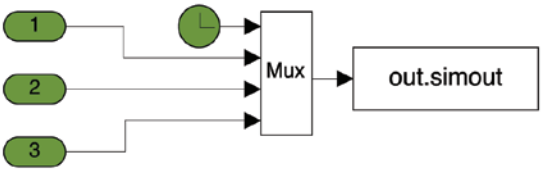
41. Ábra – Kosárlabda elkapás modell alrendszere - 1



42. Ábra - Kosárlabda elkapás modell alrendszere - 2



43. Ábra – Kosárlabda elkapás modell alrendszere - 3



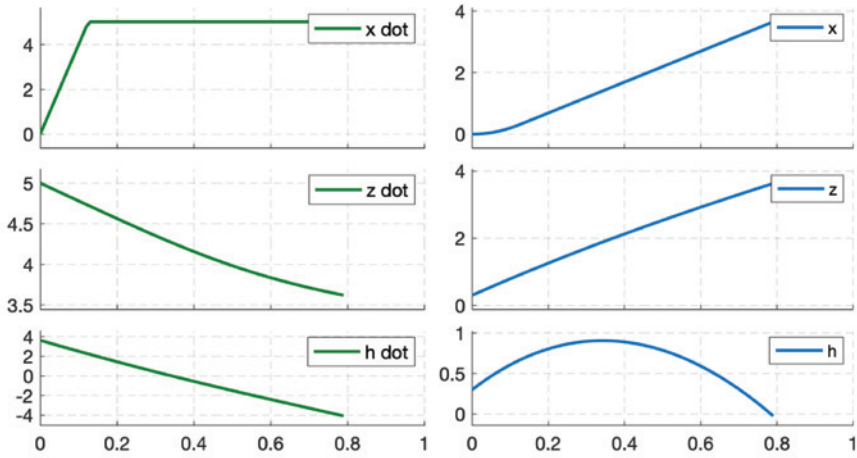
44. Ábra – Kosárlabda elkapás modell alrendszere - 4

MATLAB kód paraméterek beállítására

4. Forráskód – Paraméterek beállítása

```
clear all
clear all

F0=20;cart_mass=2;
x_dot_max= 5;
t0=0;tf=50;h=0.1;
ro_air=1.224;Cd=1;r_ball=0.15;
A_ball=pi*r_ball^2;
ball_mass=0.4;
g=-9.8;v0=5;
theta_0=pi/5
```

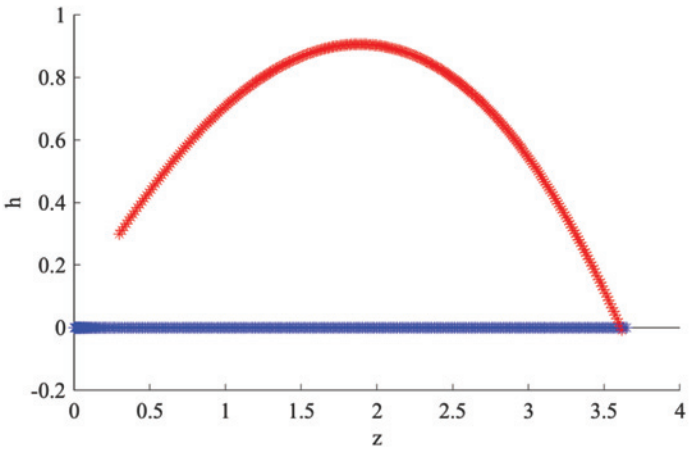
Szimuláció kimenete:



45. Ábra – Kosárlabda elkapás szimulációja

5. Forráskód - Kosárlabda elkapás vizualizáció algoritmusa

```
close all

%Read data from Simulink
t = out.simout.data(:,1);
x = out.simout.data(:,2);
z = out.simout.data(:,3);
h = out.simout.data(:,4);

hold on
%Draw default lines
plot([0 4],[0 0], 'k')
plot([0 0],[-0.2 1], 'k')

%Draw ball and cart
for i=1:length(t)
  plot(z(i),h(i), "r*")
  plot(x(i),0,  "b*")
  pause(0.1)
end
hold off
```



46. Ábra – Kosárlabda elkapás vizualizációja

2.2. Parciális differenciálegyenletek (PDE)

A PDE *(Partial Differential Equation)* többváltozós függvényeket és ezek parciális deriváltjait tartalmazó differenciálegyenlet. A PDE-k jóval bonyolultabb számításokat igényelnek az előzőekben tárgyalt ODE-khez képest, ezért gyakran számítógépen végzik a megoldásukhoz szükséges közelítő számításokat.

2.2.1. Diffúzív folytonos térmodellek

A részecskék száma az $x = a$ és $x = b$ helyzet között a következő

$$Z(t) = \int_a^b u(x, t)dx \tag{2.23}$$

A részecskék számának változásának mértéke az áramlás különbség $J(x,t)$ értékei között $x = a$ és $x = b$ esetén.

$$\frac{\partial Z(t)}{\partial t} = J(a, t) - J(b, t) \tag{2.24}$$

Az egyenletek kombinálásával a következőket kapjuk:

$$\frac{\partial}{\partial t}\int_a^b u(x, t)dx = J(a, t) - J(b, t) \tag{2.25}$$

Jól ismert, hogy az áramlás

$$J(x, t) = \frac{-d^2\, \partial u(x, t)}{\partial x}$$
$$J(a, t) - J(b, t) = \int_a^b d^2 \frac{\partial^2 (x, t)}{\partial x^2}\, dx \tag{2.26}$$

Tehát megkapjuk, hogy

$$\frac{\partial u}{\partial t} = d^2 \frac{\partial^2 u}{\partial x^2} \tag{2.27}$$

Nézzük meg az egydimenziós parabolikus differenciálegyenletet

$$\frac{\partial u}{\partial t} = d^2 \frac{\partial^2 u}{\partial x^2} + f(x, t, u) \tag{2.28}$$

ahol $a \le x \le b$ és $t_0 \le t \le t_f$ és $f(x, t)$ pedig az x helyen és t időpontban térfogategységenként létrehozott vagy megsemmisített részecskék száma. A kezdeti és peremfeltételeket a következő formában kell megadni: ha az $a \le x \le b$ és $t = t_0$, a megoldásnak egy speciális u_0 függvény esetén $u(x, t_0) = u_0(x)$ értéknek

kell megfelelnie. Az $x = a$ és $t_0 \le t \le t_f$ esetén a megoldásnak meg kell felelnie a peremfeltételnek.

Dirichlet peremfeltétel:

$$u(a, t) = g_1(t) \text{ és } u(b, t) = g_2(t) \tag{2.29}$$

Newmann peremfeltétel:

$$\frac{\partial u(a, t)}{\partial x} = \frac{\partial u(b, t)}{\partial x} = 0 \tag{2.30}$$

Robin peremfeltétel:

$$\begin{aligned} p_a(x, t, u) + q_a(x, t) \frac{\partial u(a, t)}{\partial x} &= 0 \\ p_b(x, t, u) + q_b(x, t) \frac{\partial u(b, t)}{\partial x} &= 0 \end{aligned} \tag{2.31}$$

2.2.2. Parciális differenciálegyenletek numerikus megoldása

Figyeljük meg a következő egydimenziós parabolikus differenciálegyenletet

$$\frac{\partial u}{\partial t} = d^2 \frac{\partial^2 u}{\partial x^2} + f(x, t, u) \tag{2.32}$$

ahol $a \le x \le b$ és $t_0 \le t \le t_f$. A kezdeti és peremfeltételeket a következő formában kell megadni: ha $a \le x \le b$ és $t = t_0$, a megoldásnak egy speciális u_0 függvény esetén $u(x, t_0) = u_0(x)$ értéknek kell megfelelnie. $x = a$ és $t_0 \le t \le t_f$ esetén a megoldásnak meg kell felelnie a következő feltételeknek

$$\begin{aligned} u(a, t) &= g_1(t) \\ u(b, t) &= g_2(t) \end{aligned} \tag{2.33}$$

Tegyük fel, hogy az $R = \{(x, t): a \le x \le b, t_0 \le t \le t_f\}$ téglalapot $(n-1) \times (m-1)$ téglalapokra osztjuk, amelyeknek $\Delta x = h$ és $\Delta t = k$ oldalai vannak. Kezdjük az alsó sorban, ahol $t = t_0$, a megoldás pedig $u(x_i, t_0) = u_0(x_i)$. A módszer az $u(x, t)$ közelítéseinek kiszámítására, az egymást követő sorok $\{u(x_i, t_j): i = 1, 2, ..., n\}$ rácspontjaiban, a $j = 1, 2, ...,$ m esetekben kerül kidolgozásra.
Az $u_t(x, t)$ és $u_{xx}(x, t)$ különbségképletei a következők:

$$u_t(x, t) = \frac{u(x, t + k) - u(x, t)}{k} + O(k)$$

$$u_{xx}(x, t) = \frac{u(x - h, t) - 2u(x, t) + u(x + h, t)}{h^2} + O(h^2) \quad (2.34)$$

A rácsok távolsága minden sorban egységes: $x_{i+1} = x_{i+h}$
A rácsok távolsága minden oszlopban egységes: $t_{j+1} = t_{j+k}$

Ezután elhagyjuk az $O(k)$ és $O(k^2)$ kifejezéseket, és az $u_{i,j}$ közelítést használjuk az előző egyenletekben, amelyeket behelyettesítve az eredeti egyenletbe megkapjuk a következőt:

$$\frac{u_{i,j+1} - u_{i,j}}{k} = d^2 \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} + kf_{i,j} \quad (2.35)$$

ami közelíti az eredeti egyenlet megoldását.

Az egyszerűség kedvéért bevezetjük az $r=d^2 k/h^2$ helyettesítést, és az eredmény az explicit forward-differenciálegyenlet lesz

$$u_{i,j+1} = (1 - 2r)u_{i,j} + r(u_{i-1,j} + u_{i+1,j}) + kf_{i,j} \quad (2.36)$$

Az egyenletet a $(j+1)$-edik sor létrehozására használjuk a rácson, feltételezve, hogy a j-edik sorban lévő közelítések ismertek. Megfigyelhető, hogy ez a képlet kifejezetten megadja az $u_{i,j+1}$ értéket az $u_{i-1,j}$, $u_{i,j}$ és $u_{i+1,j}$ függvényében. A képletet, egyszerűsége teszi vonzóvá. Fontos azonban, hogy olyan numerikus technikákat használjunk, amelyek stabilak. Ha a számítások szakaszában elkövetett hiba végül elenyészik, akkor a módszert stabilnak nevezzük. Az explicit forward-differenciálegyenlet akkor és csak akkor stabil, ha $r \in [0,1/2]$ intervallumra korlátozódik. Ez azt jelenti, hogy a k lépésméretnek, eleget kell tenni a $k \leq h^2/(2d^2)$ feltételnek.

6. Forráskód – Forward-differenciálegyenlet függvény

```
1    function U = forwdif (f, c1, c2, a, b, d, n, m)
2    % Bemenet - u0 = u(x, 0) mint karakterlánc 'u0'
3    % - a és b a jobb oldali végpontjai a [0,a] és [0,
     b]
4    % - d állandó a hőegyenletben
5    % - n és m rácspontok a [0, a] és [0, b] területen
6    % Kimenet- U megoldási mátrix
7    % Paraméterek és U inicializálása
8    h = a/(n-1);
9    k = b/(m-1);
```

```
10    r = d^2*k/h^2;
11    s = 1-2*r;
12    U = zeros (n, m);
13    % Peremfeltételek
14    U(1, 1:m) = c1;
15    U(n, 1:m) = c2;
16    % Első sor generálása
17    U(2:n-1, 1) = feval(f, h:h:(n-2)*h)';
18    % Az U hátralevő sorainak generálása
19    for j = 2:m
20       for i = 2:n-1
21       U(i, j) = s*U(i, j-1)+r*(U(i-1,
         j-1)+U(i+1, j-1));
22       end
23     end
24     U = U';
25    end
```

A Crank-Nicolson módszer

A John Crank és Phyllis Nicolson által kitalált implicit rendszer a

$$\frac{\partial u}{\partial t} = d^2 \frac{\partial^2 u}{\partial x^2} + f(x, t, u) \tag{2.37}$$

egyenlet megoldásának numerikus közelítésén alapul az $(x, t + k/2)$ pontban, amely a rács sorai között helyezkedik el. Konkrétan, az $u_t (x, t + k/2)$ esetében használt közelítés, a központi differencia képletből származik:

$$u_t\left(x, t + \frac{k}{2}\right) = \frac{u(x, t + k) - u(x, t)}{k} + O(k^2) \tag{2.38}$$

Az $u_{xx} (x, t + k/2)$ esetében használt közelítés az $u_{xx} (x, t)$ és $u_{xx} (x, t + k)$ közelítések átlaga, amelynek pontossága $O(h^2)$ nagyságrendű:

$$u_{xx}\left(x, t + \frac{k}{2}\right) = \frac{1}{2h^2}(u(x - h, t + k) - 2u(x, t + k) + u(x + h, t + k) + u(x - h, t) - 2u(x, t) + u(x + h, t)) + O(h^2) \tag{2.39}$$

Az előző levezetéshez hasonló módon, az értékeket behelyettesítjük az egyenletbe, és elhanyagoljuk az $O(h^2)$ és $O(k^2)$ hibatételeket. Az $u_{i,j} = u(x_j, t_j)$ jelölést alkalmazva a következő differenciálegyenletet kapjuk

$$\frac{u_{i,j+1} - u_{i,j}}{k} = d^2 \frac{u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1} + u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{2h^2} + kf_{i,j} \quad (2.40)$$

A (2.40)-ben az $r = d^2 k/h^2$ helyettesítést is alkalmazzuk. Ezúttal azonban a három „még kiszámítandó" értékre – $u_{i-1,j+1}$, $u_{i,j+1}$ és $u_{i+1,j+1}$ – kell megoldást találnunk. Ezt úgy érjük el, hogy mindegyiket az egyenlet bal oldalára helyezzük. A (2.40) egyenletben szereplő tagok átrendezése az implicit differenciálképletet eredményezi

$$-ru_{i-1,j+1} + (2 + 2r)u_{i,j+1} - ru_{i+1,j+1} = (2 - 2r)u_{i,j} + r(u_{i-1,j} + u_{i+1,j}) + kf_{i,j} \quad (2.41)$$

$i = 2, 3, ...n-1$-ig. A (2.40) egyenlet jobb oldalán lévő tagok mind ismertek. Ezért a (2.40) egyenletek egy tridiagonális lineáris rendszert alkotnak: $AX = B$. A (2.40) képletet néha az $r = 1$ arány alkalmazásával hajtják végre. Ebben az esetben a t tengely mentén a növekmény $\Delta t = k = h^2/d^2$, és így a (2.41) egyenletek is egyszerűsödnek, a következőképpen:

$$-u_{i-1,j+1} + 4u_{i,j+1} - u_{i+1,j+1} = u_{i-1,j} + u_{i+1,j} + kfs_{i,j} \quad (2.42)$$

$i = 2, 3, ...n-1$-ig. A peremfeltételeket az első és az utolsó egyenletben használjuk, azaz $u_{1,j} + u_{1,j+1} = c_1$ és $u_{n,j+1} = c_2$. A (2.42) egyenletet különösen tetszetős a tridiagonális mátrixa $AX = B$-től.

$$\begin{bmatrix} 4 & -1 & & & & \\ -1 & 4 & -1 & & 0 & \\ & \ddots & \ddots & \ddots & & \\ & & -1 & 4 & -1 & \\ & & & \ddots & \ddots & \ddots \\ & 0 & & & -1 & 4 & -1 \\ & & & & & -1 & 4 \end{bmatrix} \begin{bmatrix} u_{2,j+1} \\ u_{3,j+1} \\ \vdots \\ u_{p,j+1} \\ \vdots \\ u_{n-2,j+1} \\ u_{n-1,j+1} \end{bmatrix} = \begin{bmatrix} 2c_1 + u_{3,j} + kf_{2,j} \\ u_{2,j} + u_{4,j} + kf_{3,j} \\ \vdots \\ u_{p-1,j} + u_{p+1,j} + kf_{p,j} \\ \vdots \\ u_{n-3,j} + u_{n-1,j} + kf_{n-2,j} \\ u_{n-2,j} + 2c_2 + kf_{n-1,j} \end{bmatrix}. \quad (2.43)$$

A Crank-Nicolson-módszer számítógépes megvalósításával, az $AX=B$ lineáris rendszer közvetlen módszerekkel, vagy iteratív eljárásokkal megoldható.

7. Forráskód – Crank-Nicolson-módszer függvénye

```
1    function U = crnich (f, c1, c2, a, b, d, n, m)
2    % Bemenet - f = u(x, 0) mint 'f' karakterlánc
3     % - c1 = u(0, t) és c2 = u(a, t)
4    % - a és b a jobb oldali végpontjai a [0, a] és [0, b]
5    % - d állandó a heat equation
6    % - n és m rácspontok a [0, a] és [0, b] területen
7    % Kimenet - U megoldás mátrix
```

```
8     % Paraméterek és U inicializálása
9
10    h = a/(n-1);
11    k = b/(m-1);
12    r = d^2*k/h^2;
13    s1 = 2+2/r;
14    s2 = 2/r-2;
15    U = zeros (n, m);
16    % Peremfeltételek
17    U(1, 1:m) = c1;
18    U(n, 1:m) = c2;
19    % Első sor generálása
20    U(2:n-1, 1) = feval(f, h:h:(n-2)*h) ';
21     % Megalkotja a főátlót és a főátlón kívüli elemeket
22    % az A és a konstans B vector segítségével majd
      megoldja
23     % az AX = B tridiagonális rendszert
24    Vd(1, 1:n) = s1*ones(1, n);
25    Vd(1) = 1;
26    Vd(n) = 1;
27    Va = -ones(1, n-1);
28    Va(n-1) = 0;
29    Vc = -ones(1, n-1);
30
31    Vc(1) = 0;
32    Vb(1) = c1;
33    Vb(n) = c2;
34    for j = 2:m
35       for i = 2:n-1
36       Vb(i) = U(i-1, j-1)+U(i+1, j-1)+s2*U(i, j-1);
37       end
38       X = trisys(Va, Vd, Vc, Vb);
39       U(1:n, j) = X' ;
40     end
41     U = U';
42    end
```

Féldiszkretizáló módszerek parciális differenciálegyenletekhez
Vessünk egy pillantást a következő PDE-re

$$\frac{\partial}{\partial t}u(x,t) + = d^2 \frac{\partial^2}{\partial x^2} u(x,t) + f(x,t,u),\ a \le x \le b \tag{2.44}$$

Dirichlet peremfeltételekkel $u(a,t) = g_1(t)$, $u(b,t) = g_2(t)$, és kezdeti adatokkal $u(x,0) = u_0(x)$, a rácsháló távolsága minden sorban egységes:

$$x_{i+1} = x_i + h,\ i = 0, \ldots, n-1,\quad h = (b-a)/n \tag{2.45}$$

Jelölése

$$U(t) = (u(a,t), u(x_1,t), \ldots, u(x_{n-1},t), u(b,t)) = (u_0(t), \ldots, u_n(t)) \tag{2.46}$$

A térderiváltak diszkretizálásával a következő ODE-ket kapjuk:

$$\begin{aligned} \dot{u}_1(t) &= r(u_0(t) - 2u_1(t) + u_2(t)) + f(x_1, t, u_1) \\ \dot{u}_2(t) &= r(u_1(t) - 2u_2(t) + u_3(t)) + f(x_2, t, u_2) \\ &\vdots \\ \dot{u}_{n-1}(t) &= r(u_{n-2}(t) - 2u_{n-1}(t) + u_n(t)) + f(x_{n-1}, t, u_{n-1}) \end{aligned} \tag{2.47}$$

Mátrix alakban:

$$\dot{u}(t) = Au(t) + v + f(x,t,u) \tag{2.48}$$

ahol:
A mátrix $N-1 \times N-1$ és $u(t) = (u_1(t), \ldots, u_{n-1}(t)$

$$A = r \begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & 0 & \\ & & \ddots & & & & \\ & & 1 & -2 & 1 & & \\ & & & & \ddots & & \\ & 0 & & & 1 & -2 & 1 \\ & & & & & 1 & -2 \end{bmatrix}, \tag{2.49}$$

ahol $r = d^2/h^2$ és $v = (ru_0(t), 0, \ldots, 0, ru_n(t))$. Itt a, b és $h = 1/n$ paraméterek. Ez az ODE-rendszer akkor keletkezik, ha a központi differenciákon alapuló vonalak módszerét használjuk a parciális differenciálegyenlet (PDE) féldiszkretizálására.

2.2.a. Példa – Fél-diszkretizáló módszerek parciális differenciálegyenletekhez

$$\begin{aligned} &\frac{\partial}{\partial t} u(x,t) + = d^2 \frac{\partial^2}{\partial x^2} u(x,t) + u(x,t)(1 - u(x,t)) \\ &0 \le x \le 1 \end{aligned} \tag{2.50}$$

Dirichlet peremfeltételekkel $u(0, t) = u(1, t) = 0$ és $u(x, 0) = (1+\cos(2\pi x))/2$ kezdeti feltételekkel

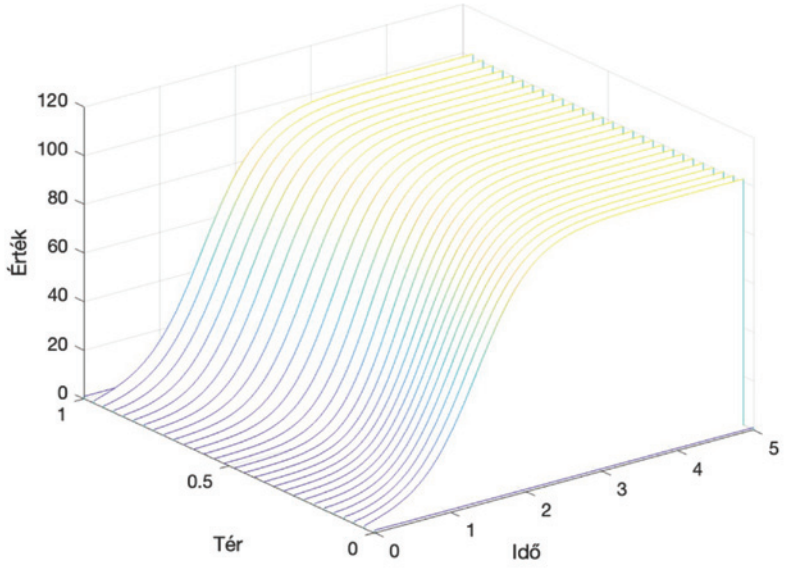
8. Forráskód – rcd függvény

```
1     function rcd
2       %RCD Stiff ODE from method of lines on reaction-
        diffusion
3       %problem.
4       N = 20; a = 1;h=1/(N-1);d=.01;r=3;K=100;
5       A=zeros(N,N);
6       A(2,1)=1;
7       A(N-1,N)=1;
8       for i=2:N-1
9       A(i,i)=-2;
10      end
11      for i=2:N-2
12      A(i+1,i)=1;
13      A(i,i+1)=1;
14      end
15      A=d/h^2*A;
16      tspace =[0 5]; space = [1:N]/(N-1);
17      y0 = 0.5*(1+cos(2*pi*space));
18      y0 = y0(:); y0(1)=0;y0(N)=0; %Dirichlet
        peremfeltétel
19      [t,y] = ode45(@f,tspace,y0);
20      waterfall(t,[0:1/(N-1):1],y')
21      xlabel('Idő','FontSize',16),
22      ylabel('Tér','FontSize',16)
23       % ---------------------- Beágyazott függvények
        -----------
24      function dydt = f(t,y)
25      %F  differenciál függvény.
26      dydt = A*y+r*y.*(1-y/K);
27      dydt(1)=0;dydt(N)=0;
28      end
29    end
30
```

9. Forráskód – rcd_Neumann függvény

```
function rcd_Neumann
 %RCD Stiff ODE from method of lines on reaction-convection-
 diff
 %problem.
 N = 30; a = 1;h=1/(N-1);d=.01;r=3;K=100;
 A=zeros(N,N);
 A(2,1)=1;A(2,2)=-1;
 A(N-1,N)=1;A(N-1,N-1)=-1;
 for i=3:N-2
   A(i,i)=-2;
 end
 for i=2:N-2
 A(i+1,i)=1;
 A(i,i+1)=1;
 end
 A=d/h^2*A;
 tspace =[0 5]; space = [1:N]/(N-1);
 y0 = 0.5*(1+cos(2*pi*space));
 y0 = y0(:);
  [t,y] = ode45(@f,tspace,y0);
 waterfall(t,[0:1/(N-1):1],y')
 xlabel('Idő', 'FontSize',16),
 ylabel('Tér', 'FontSize',16),
 zlabel('Érték', 'FontSize',16)
 % -------------------- Beágyazott függvények ----------
 function dydt = f(t,y)
 %F     differenciál függvény.
 dydt = A*y+r*y.*(1-y/K);

 dydt(1)=0;dydt(N)=0;
 end
end
```

Szimuláció kimenete:



47. Ábra – RCD függvény numerikus megoldása

2.2.3. Részleges differenciálegyenletek pdepe segítségével

A MATLAB pdepe a parabolikus/elliptikus differenciálegyenlet (PDE) rendszerek egy osztálya. Ezek a rendszerek egy vektorértékű ismeretlen u függvényt tartalmaznak, amely függ az x skaláris térváltozótól és a t skaláris időváltozótól. A pdepe által használt általános osztály az alábbi formájú:

$$c\left(x, t, u, \frac{\partial u}{\partial x}\right) \frac{\partial u}{\partial t} = x^m \frac{\partial}{\partial x}\left(x^m F\left(x, t, u, \frac{\partial u}{\partial x}\right)\right) + f\left(x, t, u, \frac{\partial u}{\partial x}\right) \qquad (2.51)$$

ahol $a \le x \le b$ és $t_0 \le t \le t_f$. Az m egész szám lehet 0, 1 vagy 2, ami megfelel a lemez-, a henger- és a gömbszimmetriának. A c függvény egy diagonális mátrix, az átáramlás és a forrásfüggvény – f és s – pedig vektorértékű.
A kezdeti és peremfeltételeket a következő formában kell megadni: $a \le x \le b$ és $t = t_0$ esetén a megoldásnak meg kell felelnie $u(x, t_0) = u_0(x)$-nek, vagy egy meghatározott u_0 függvénynek. $x = a$ és $t_0 \le t \le t_f$ esetén, a megoldásnak meg kell felelnie a következő feltételeknek

$$p_a(x, t, u) + q_a(x, t) F\left(x, t, u, \frac{\partial u}{\partial x}\right) = 0 \qquad (2.52)$$

valamilyen p_a és q_a függvényekre. Hasonlóképpen, $x = b$ és $t_0 \le t \le t_f$ esetén,

$$p_b(x, t, u) + q_b(x, t)F\left(x, t, u, \frac{\partial u}{\partial x}\right) = 0 \tag{2.53}$$

a p_b és q_b függvényekre kell vonatkoznia. A pdepe által megoldható problémák osztályára bizonyos egyéb korlátozások is vonatkoznak; a részleteket lásd a doc pdepe[1] dokumentumban.

A *pdepe* hívása a következő általános formában történik

sol=pdepe(m, @pdefun, @pdeic, @pdebc, xmesh, tspan, options, p1, p2,...),

ami hasonló a *bvp4c* szintaxisához. Az *m* bemeneti argumentum a fent leírtak szerint 0,1 vagy 2 értéket vehet fel. A pdefun függvény a következő formájú

function[c,f,s] = pdefun(x,t,u,DuDx,p1,p2,...)

A pdefun függvény a tér- és időváltozókat, valamint az *u* és *DuDx* vektorokat használja, amelyek közelítik az *u* megoldást és az *∂u/∂x* parciális deriváltat. Továbbá a függvény visszaadja a *c* mátrix diagonálisát, valamint az *f* áramlás és az *s* forrásfüggvényeket tartalmazó vektorokat. A kezdeti feltételek a pdeic függvényben vannak kódolva, amely a következő formájú:

function u0 = pdeic(x, p1, p2,...)

A pdebc függvény az alábbi formában

function [pa,qa,pb,qb] = pdebc(xa, ua, xb, ub, t, p1, p2,...)

kiértékeli *pa, qa, pb* és *qb* értékeit, az *xa = a* és *xb = b* peremfeltételekhez. A pdepe argumentumlistájában szereplő *xmesh* vektor az *[a,b]* pontok halmaza, ahol *xmesh(1) = a* és *xmesh(end) = b*, úgy rendezve, hogy *xmesh(i) < xmesh (i + 1)*. Ez határozza meg azokat az *x* értékeket, amelyeknél a numerikus megoldás kiszámításra kerül. Az algoritmus az *xmesh* értékek alapján másodrendű térbeli diszkretizációs módszert használ, ezért az *xmesh* értékek megválasztása erősen befolyásolja a numerikus megoldás pontosságát és költségeit. Szorosan elhelyezkedő *xmesh* pontokat kell használni azokban

[1] https://www.mathworks.com/help/matlab/ref/pdepe.html (2024.08.02.)

a régiókban, ahol a megoldások valószínűleg gyorsan változnak *x* tekintetében. A *tspan* vektor megadja azokat az időpontokat a $[t_0, t_f]$ tartományban, ahol a megoldást vissza kell adni: $tspan(1) = t_0$, $tspan(end) = t_f$ és $tspan(i) < tspan(i+1)$. Az időintegrációt a pdepe-ben az ode45s végzi, a tényleges időlépések értékeit dinamikusan választjuk ki. A tspan pontok egyszerűen meghatározzák, hogy hol adjuk vissza a megoldást, és kevés hatással vannak a költségekre vagy a pontosságra.

Példák

A következőkben tekintsünk meg néhány gyakorlati alkalmazást.

2.2.b. Példa - Diffúziós kompetíciós rendszer

Elsőként figyeljük meg a következő diffúziós kompetíciós rendszert

$$\begin{aligned}
\frac{\partial}{\partial(t)} y_1(x,t) &= D_1 \frac{\partial^2}{\partial x^2} y_1(x,t) + y_1(a_1 - a_2 y_2(x,t) - a_3 y_3(x,t)) \\
\frac{\partial}{\partial(t)} y_2(x,t) &= D_2 \frac{\partial^2}{\partial x^2} y_2(x,t) + y_2(a_4 - a_5 y_3(x,t)) \\
\frac{\partial}{\partial(t)} y_3(x,t) &= D_3 \frac{\partial^2}{\partial x^2} y_3(x,t) + y_3(-a_8 - a_6 y_1(x,t) - a_7 y_2(x,t))
\end{aligned} \tag{2.54}$$

Neumann peremfeltétellel

$$\frac{\partial}{\partial x} y_i(x,t) = 0, \qquad x = 0, \qquad \pi, t \geq 0 \tag{2.55}$$

és az alábbi kezdeti feltételek mellett

$$y_i(x,0) = \Phi_i(x) \geq 0,\ 0 \leq x \leq \pi \tag{2.56}$$

ahol D_i, a_i mind pozitív konstans.

Legyen $\bar{x} = \left(\bar{x}_3 = \frac{a_4}{a_5},\ \bar{x}_2 = \frac{a_1 - a_3 \bar{x}_3}{a_2},\ \bar{x}_1 = \frac{a_8 - a_7 \bar{x}_3}{a_6}\right)$

Ha $x_i > 0$, akkor van egy pozitív térbeli, homogén egyensúlyi pont és az ezen egyensúly körüli linearizáció a következő formát ölti

$$\begin{aligned}
\frac{\partial}{\partial t} y_1(x,t) &= D_1 \frac{\partial^2}{\partial x^2} y_1(x,t) + \bar{y_1}(-a_2 y_2(x,t) - a_3 y_3(x,t)) \\
\frac{\partial}{\partial t} y_2(x,t) &= D_2 \frac{\partial^2}{\partial x^2} y_2(x,t) + \bar{y}_2(-a_5 y_3(x,t)) \\
\frac{\partial}{\partial t} y_3(x,t) &= D_3 \frac{\partial^2}{\partial x^2} y_3(x,t) + \bar{y}_3(a_6 y_1(x,t) + a_7 y_2(x,t))
\end{aligned} \tag{2.57}$$

Legyen $X = (C[0,\pi]; R^3)$. Mivel a Laplace-operátornak $-k^2$ sajátértékei vannak, $k = 0, 1...$ a megfelelő *coskx* sajátfüggvényekkel, λ a linearizáció karakterisztikus értéke, ha bizonyos $k = 0,1$ esetén λ a karakterisztikus egyenlet megoldása.

$$\lambda v = (-k_2 D + J) v, \qquad k = 0, 1, \tag{2.58}$$

ahol J egy Jacobi-mátrix és D egy diagonális mátrix, ahol $d_{ii}=d$. A linearizálás-ból kapott karakterisztikus egyenlet a következő:

$$\lambda^3 + A_1(k)\lambda^2 + A_2(k)\lambda + A_3(k) = 0,$$

ahol:

$A_1(k) = k^2 (d_1 + d_2 + d_3)$

$A_2(k) = k^4 (d_2 d_3 + d_1 (d_1 + d_2)) + a_3 a_6 \bar{x}_1 \bar{x}_3 + a_5 a_7 \bar{x}_2 \bar{x}_3$

$A_3(k) = k^6 d_1 d_2 d_3 + n^2 (d_2 a_3 a_6 \bar{x}_1 \bar{x}_3 + d_1 a_5 a_7 \bar{x}_2 \bar{x}_3) - a_2 a_5 a_6 \bar{x}_1 \bar{x}_2 \bar{x}_3)$

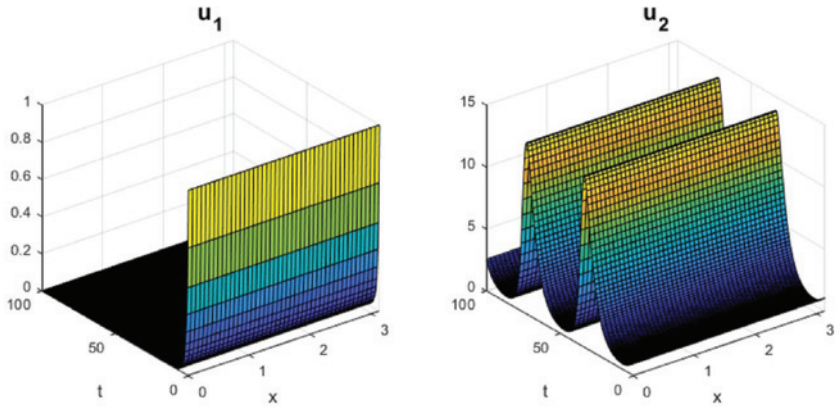
A stabilitás vizsgálatához a Routh-Hurwitz-kritériumot használjuk. $k = 0$ esetén $A_1(k) = 0$, ami az $\bar{x}$ egyensúlyi pont instabilitását jelenti. A következő MATLAB kód a rendszer numerikus megoldását adja.

MATLAB kód:

10. Forráskód – Diffúziós kompetíció rendszer

```
% Reaction-diffusion competition system
% Solves the PDE
global a1 a2 a3 a4 a5 a6 a7 a8 d1 d2 d3
a2=0.93; a3=0.1;
a4=0.19; a5=0.2;
a6=1; a7=0.05;
a8=0.2; d1=0.01;
d2=0.03; d3=0.009;
%a1=1.01;
a1=0.1;
m=0;
xmesh = linspace(0,3.14,45);
tspan = linspace(0,100,250);
sol = pdepe(m,@mbpde,@mbic,@mbbc,xmesh,tspan);
u1 = sol(:,:,1);
u2 = sol(:,:,2);
subplot(121)
```

```
18      surf(xmesh,tspan,u1)
19      xlabel('x', 'Fontsize', 12)
20      ylabel('t','Fontsize', 12)
21      title('u_1','FontSize', 16)
22      subplot(122)
23      surf(xmesh,tspan,u2)
24
25      xlabel('x','Fontsize', 12)
26      ylabel('t','Fontsize', 12)
27      title('u_2','FontSize', 16)
28      % ------------------------------------------------
        -----------
29      % Alfüggvények
30      % ------------------------------------------------
        -----------
31      function [c,f,s] = mbpde(x,t,u,DuDx)
32        global a1 a2 a3 a4 a5 a6 a7 a8 d1 d2 d3
33        c = [1; 1; 1];
34        f = [d1; d2; d3] .* DuDx;
35        s = [u(1)*(a1-a2*u(2)-a3*u(3)); u(2)*(a4-
          a5*u(3)); u(3)*(-
36      a8+a6*u(1)+a7*u(2))];
37      end
38      % ------------------------------------------------
        -----------
39      function u0 = mbic(x)
40        u0 = [1; 1; 1];
41      end
42      % ------------------------------------------------
        ----------
43      function [pa,qa,pb,qb] = mbbc(xl,ul,xr,ur,t)
44        pa = [0;0;0];
45        qa = [1;1;1];
46        pb = [0;0;0];
47        qb = [1;1;1];
48      end
```



48. Ábra – Diffúziós kompetíció rendszer kimenete

2.2.c. Példa – A kemosztát diffúziós matematikai modellje

Egy másik alternatíva egyszerűen egy edény használata és az alap kemosztát „jól felkavart" hipotézisének elhagyása, ami a következő formájú reakció-diffúziós egyenletrendszert eredményezi egyenlő diffúziós sebességet feltételezve:

$$\begin{aligned} \frac{\partial S}{\partial t} &= d\,\frac{\partial^2 S}{\partial x^2} - \frac{m_1 Su}{a_1 + S} - \frac{m_2 Sv}{a_2 + S} \\ \frac{\partial u}{\partial t} &= d\,\frac{\partial^2 u}{\partial x^2} + \frac{m_1 Su}{a_1 + S} \\ \frac{\partial v}{\partial t} &= d\,\frac{\partial^2 v}{\partial x^2} - \frac{m_2 Sv}{a_2 + S}, \; 0 < x < \pi \end{aligned} \tag{2.59}$$

az alábbi peremfeltételekkel:

$$\begin{aligned} \frac{\partial S}{\partial x}(t, 0) &= -S^{(0)} \\ \frac{\partial u}{\partial x}(t, 0) &= \frac{\partial v}{\partial x}(t, 0) = 0 \\ \frac{\partial S}{\partial x}(t, \pi) + rS(t, \pi) &= 0 \\ \frac{\partial u}{\partial x}(t, \pi) + ru(t, \pi) &= 0 \\ \frac{\partial v}{\partial x}(t, \pi) + rv(t, \pi) &= 0 \end{aligned} \tag{2.60}$$

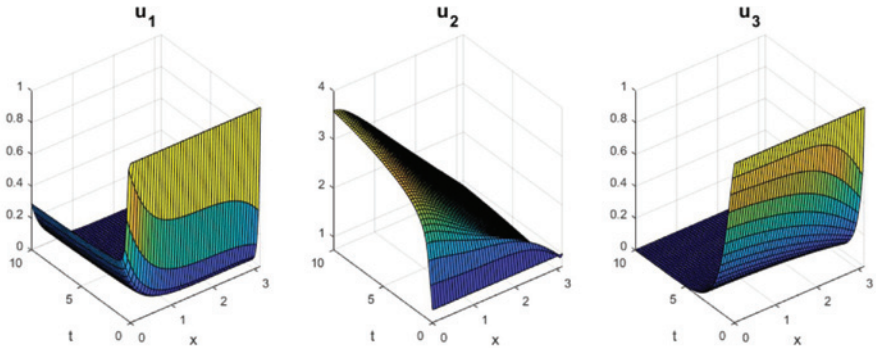
és az alábbi kezdeti értékekkel:

$$S(0, x) = S_0(x) \geq 0$$
$$u(0, x) = u_0(x) \geq 0, u_0(x) \neq 0 \qquad (2.61)$$
$$v(0, x) = v_0(x) \geq 0, v_0(x) \neq 0$$

11. Forráskód – A kemosztát diffúziós matematikai modellje

```
function chemostat
% Reakció-diffúziós kemosztát rendszer
% PDE megoldására
global a1 a2 m1 m2 d1 d2 d3 s0
d1=1;d2=1; d3=1;s0=1.;
a1=1; a2=1; m1=4; m2=1;
m=0;
xmesh = linspace(0,3.14,45);
tspan = linspace(0,10,50);
sol = pdepe(m,@mbpde,@mbic,@mbbc,xmesh,tspan);
u1 = sol(:,:,1);
u2 = sol(:,:,2);
u3 = sol(:,:,3);
subplot(131)
surf(xmesh,tspan,u1)
xlabel('x', 'Fontsize', 12)
ylabel('t', 'Fontsize', 12)
title('u_1', 'FontSize ' , 16)
subplot(132)
surf(xmesh,tspan,u2)
xlabel('x', 'Fontsize', 12)
ylabel('t', 'Fontsize', 12)
title('u_2', 'FontSize', 16)
subplot(133)
surf(xmesh,tspan,u3)
xlabel('x', 'Fontsize', 12)
ylabel('t', 'Fontsize', 12)
title('u_3', 'FontSize', 16)
end

% --------------------Alfüggvények
```

```
32          function [c,f,s] = mbpde(x,t,u,DuDx)
33          global a1 a2 m1 m2 d1 d2 d3 s0
34          c=[1;1;1];
35          f = [d1;d2;d3].*DuDx;
36          s    =    [-m1*u(1)*u(2)/(a1+u(1))-m2*u(1)*u(3)/
            (a2+u(1));
37          m1*u(1)*u(2)/(a1+u(1));          m2*u(1)*u(3)/
            (a2+u(1))-s0*u(3)];
38          end
39
40          function u0 = mbic(x)
41          u0 = [1;1;1];
42          end
43
44          function [pa,qa,pb,qb] = mbbc(xa,ua,xb,ub,t)
45          global a1 a2 m1 m2 d1 d2 d3 s0
46          pa = [s0;0;0];
47          qa = [1;1;1];
48          pb = [ub(1);ub(2);ub(3)];
49          qb = [1;1;1];
50          end
```



49. Ábra – Kemosztát kimenete

2.2.4. Késleltetési differenciálegyenlet numerikus megoldása - Lineáris spline közelítés

Bemutatunk egy hasznos technikát a differencia-differenciálegyenletek közelítésére a következő formában

$$\dot{x}(t) = f(x(t), x(t-\tau)) \tag{2.62}$$

azzal a céllal, hogy numerikusan megoldjuk.

Az ötlet az, hogy a τ hosszúságú késleltetési intervallumot N intervallumra osztjuk. A felosztás minden egyes t_j pontján definiáljuk, hogy x_{tj} egy lineáris spline csomópontja. Az $x(t)$ függvényt ezen az intervallumon egy darabonkénti lineáris függvénnyel fogjuk közelíteni. A közelítő függvényt az $(y_0,...y_N)$ függvénnyel írjuk le. Legyen

$$y_j(t) = x(t - j\tau/N) \tag{2.63}$$

az $N+1$ $j=0,...,N$ csomópontra. Legyen $yN(t) = (t-\tau)$. A csomópontok között y_j legyen egy lineáris függvény. Látható, hogy az $y_j(t)$ megközelítőleg

$$y_j(t) = \frac{(y_{j-1} - y_j)}{\tau/N} \tag{2.64}$$

A lineáris spline közelítést alkalmazva megkapjuk a közönséges differenciálegyenletek rendszerét (amely nem tartalmaz időkésleltetést)

$$\begin{aligned} \dot{y}_0(t) &= f(y_0, y_N) \\ \dot{y}_j(t) &= \frac{N}{\tau}(y_{j-1} - y_j) \end{aligned} \tag{2.65}$$

$j=1, ..., N$ esetén. Ezzel a jelöléssel, a fent felsorolt feltételek mellett, a következő tételt kapjuk:

$$x(t-\tau) = y_N(t) + O(1/N) \tag{2.66}$$

mint $N\to\infty$, egységesen t-re bármely véges halmazon, amelyen $x(t)$ létezik.

A DDE megoldó képes megoldani a közönséges differenciálegyenletek rendszereit

$$\dot{x}(t) = f(t, y(t), y(t-\tau_1), ..., y(t-\tau_k)) \tag{2.67}$$

ahol t a független változó, y a függő változó, és $\dot{y}$ a dy/dt-t reprezentálja. A késleltetések (lags) $\tau_1, ..., \tau_k$ pozitív konstansok. A kezdeti értékproblémában

a megoldást egy $[t_0, t_f]$ intervallumon keressük, ahol $t_0 < t_f$. A DDE azt mutatja, hogy $\dot{y}(t)$ függ a megoldás t előtti időpontokban mért értékeitől. Különösen $\dot{y}(t_0)$ függ a $t \leq t_0$ értékeitől, vagyis az $S(t)$ előzményétől.

A dde23 függvény állandó késleltetésű késleltetési differenciálegyenletek (DDE-k) kezdeti értékproblémáit oldja meg. Ez integrál egy elsőrendű differenciálegyenlet-rendszert

$$\dot{y}(t) = f(t, y(t), y(t - \tau_1), \ldots, y(t - \tau_k)) \tag{2.68}$$

a $[t_0, t_f]$ intervallumon, ahol $t_0 < t_f$, és adott előzmény $y(t) = S(t)$ a következő esetekben $t \leq t_0$. A dde23 olyan megoldást ad, amely folytonos a $[t_0, t_f]$ tartományban. A *deval*[2] függvényt és a *dde23* kimenetét használhatja a megoldás kiértékelésére az integrálási intervallum meghatározott pontjain.

A *dde23* követi a diszkontinuitásokat, és integrálja a differenciálegyenleteket az *ode23* által használt explicit Runge-Kutta (2,3) párral és interpolánssal.

A DDE megoldó alapvető szintaxisa a következő:

sol = dde23(ddefun,lags,history,tspan)[3]

A bemeneti argumentumok pedig a

ddefun()

a differenciálegyenletek jobb oldalát kiértékelő függvény. A függvénynek a következő formájúnak kell lennie

dydt=ddefun(t,y,z)

ahol a skalár t a független változó, az oszlopvektor y a függő változó, és $\tau_j = lags(j)$ esetén $Z(:, j) = y(t - \tau_j)$.

lags

állandó pozitív $\tau_1,\ldots,\tau_k$ késleltetések vektora.

history()

[2] https://www.mathworks.com/help/matlab/ref/deval.html (2024.08.02.)
[3] https://www.mathworks.com/help/matlab/ref/dde23.html (2024.08.02.)

a t függvény, amely kiértékeli az $y(t)$ megoldást $t \le t_0$ esetén. A függvénynek a következő formájúnak kell lennie

$$S = history(t)$$

ahol S egy oszlopvektor. Alternatívaként, ha $y(t)$ konstans, megadhatjuk a history-t is, mint ezt a konstans vektort. Ha a dde23 aktuális hívása egy korábbi integrálást folytat t_0-ig, akkor az adott hívásból származó megoldási *sol*-t használja előzményként.

$$tspan$$

az integrációs intervallum, ami egy kételemű vektor $[t_0, t_f]$, ahol $t_0 < t_f$ késleltetett előzmények.

A sol kimeneti argumentuma a solver által létrehozott struktúra. Ennek a következő mezői vannak:

sol.x – A rács dde23 által kiválasztott csomópontjai
sol.y – Az y(t) közelítése a sol.x hálópontjaiban
sol.yp – Az $\dot{y}(t)$ közelítése a sol.x hálópontjaiban
sol.solver – 'dde23'

A numerikus megoldás kiértékeléséhez a $[t_0, t_f]$ intervallum bármely pontján a *deval*-t használjuk, ahol bemenetként a sol struktúrát adjuk meg.

Fejlettebb alkalmazásokhoz bemeneti argumentumként megadhatunk megoldási opciókat és további paramétereket is.

2.2.d. Példa - Konstans késleltetésű DDE-rendszer

Ez a példa egy konstans késleltetésű DDE-rendszer megoldásának egyszerű megfogalmazását, kiszámítását és megjelenítését mutatja be. Az előzmény állandó, ami gyakran előfordul. A differenciális egyenletek a következők

$$\begin{aligned} \dot{y}_1(t) &= y_1(t-1) \\ \dot{y}_2(t) &= y_1(t-1) + y_2(t-0.2) \\ \dot{y}_3(t) &= y_2(t) \end{aligned} \tag{2.69}$$

A példa a [0,5] egyenleteket oldja meg előzményekkel.

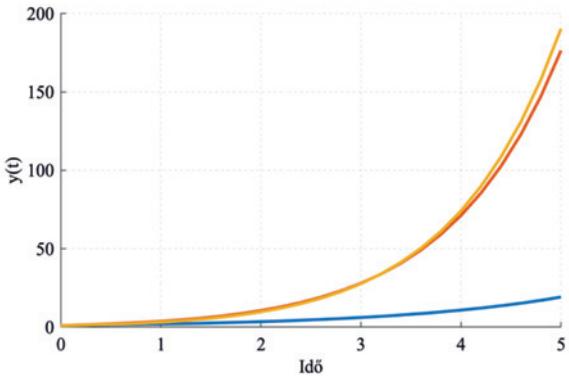
$$\begin{aligned} y_1(t) &= 1 \\ y_2(t) &= 1, \text{ ahol } t \le 0 \\ y_3(t) &= 1 \end{aligned} \tag{2.70}$$

MATLAB kód:

12. Forráskód – Konstans késleltetésű DDE-rendszer forráskódja

```
clear all, close all

sol = dde23(@ddex1de,[1, 0.2],@ddex1hist,[0, 5]);
plot(sol.x,sol.y)

title('Wille 'és Baker példa');
xlabel('Idő');
ylabel('y(t)');

function s = ddex1hist(t)
s = ones(3,1);
end

function dydt = ddex1de(t,y,Z)
ylag1 = Z(:,1);
ylag2 = Z(:,2);
dydt = [ ylag1(1)
ylag1(1) + ylag2(2)
y(2) ];
end
```

A szimuláció kimenete:



50. Ábra – Konstans késleltetésű DDE-rendszer szimulációjának kimenete

2.2.e. Példa - Zsákmány-ragadozó rendszer diffúzióval

Tekintsük meg a következő zsákmány-ragadozó rendszert diffúzióval és diszkrét időbeli késleltetéssel

$$\frac{\partial}{\partial t} u_1(x,t) = D_1 \frac{\partial^2}{\partial x^2} u_1(x,t)(1 - u_1(x,t-\tau) - u_2(x,t))$$

$$\frac{\partial}{\partial t} u_2(x,t) = D_2 \frac{\partial^2}{\partial x^2} u_2(x,t) + u_2(x,t)(1 - u_2(x,t-\tau) + u_1(x,t) + u_3(x,t))$$

$$\frac{\partial u_3}{\partial t}(x,t) = D_2 \frac{\partial^2}{\partial x^2} u_3(x,t) + u_3(x,t)(1 - u_3(x,t-\tau) - u_2(x,t)) \qquad (2.71)$$

Neumann peremfeltétellel

$$\frac{\partial}{\partial x} u_i(x,t) = 0, \quad x = 0, \quad \pi, t \geq 0 \qquad (2.72)$$

és az alábbi kezdeti feltételekkel

$$u_i(x,t) = \Phi(x,t) \geq 0, 0 \leq x \leq \pi, t \in \langle -\tau, 0 \rangle \qquad (2.73)$$

ahol D_1, D_2, d_3, a, b, c, τ, β pozitív konstansok

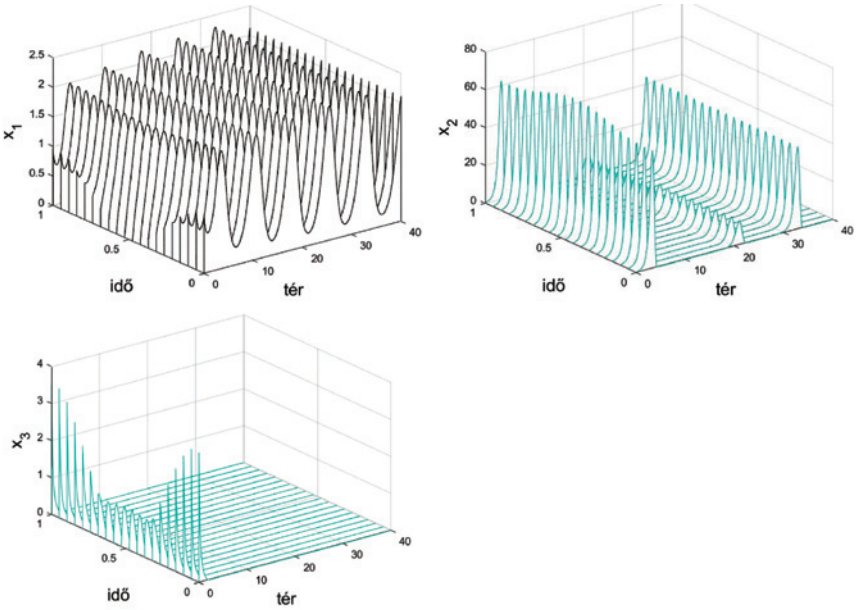
MATLAB kód:

13. Forráskód – Zsákmány-ragadozó rendszer diffúzióval forráskódja

```
1       %PDE model carasius with time delay
2       function PDE_Delay
3       N = 18; a = 1; b = 5e-2;
4       tspan = [0 40.]; space = [1:N]/(N+1);
5       x10 = 0.5*(1+cos(2*pi*space));
6       x20 = 0.1*(1+cos(2*pi*space));
7       x30 = 1.8*(1+cos(2*pi*space));
8
9        y0(1:N) = x10(:);
10      y0(N+1:2*N) = x20(:); y0(2*N+1:3*N) = x30(:);
11       y0 = y0';
12      sol = dde23(@f,1.8,y0,tspan);
13      e = zeros(size(sol.x))';
14      U1 = [sol.y(1,:)'sol.y(1:N,:)'sol.y(N,:)'];
15      U2 = [sol.y(N+1,:)'sol.y(N+1:2*N,:)'sol.y(2*N,:)'
        ];
16      U3 = [sol.y(2*N+1,:)'sol.y(2*N+1:3*N,:)'
        sol.y(3*N,:)'];
```

```
 [n m] = size(U1);
colormap([0 0 0])
 C = ones(n,m);

 figure(1)
waterfall(sol.x,[0:1/(N+1):1],U1',C')
xlabel( 'space', 'FontSize',16),
ylabel( 'time', 'FontSize',16)
zlabel( 'x_1', 'FontSize ' ,16)
 figure(2)
waterfall(sol.x,[0:1/(N+1):1],U2',C')
xlabel( 'space', 'FontSize',16)
ylabel( 'time', 'FontSize',16)
zlabel( 'x_2', 'FontSize',16)
 figure(3)
waterfall(sol.x,[0:1/(N+1):1],U3',C')
xlabel( 'space', 'FontSize',16)
 label( 'time', 'FontSize',16)
zlabel( 'x_3', 'FontSize',16)

function dydt = f(t,y,Z)
r2 = b*(N+1)^2;
 up1 = [y(2:N);0]; down1 = [0;y(1:N-1)];
up2 = [y(N+2:2*N);0]; down2 = [0;y(N+1:2*N-1)];
up3 = [y(2*N+2:3*N);0]; down3 =
[0;y(2*N+1:3*N-1)];
ylag = Z(:,1);
 mat = -2*eye(N,N);mat(1,1)=-1;mat(N,N)=-1;
dydt(1:N) = r2*(mat*y(1:N)+up1+down1)+
y(1:N).*(1-ylag(1:N));
dydt(N+1:2*N) = r2*(mat*y(N+1:2*N)+up2+down2)+y(N
+1:2*N).*(1-ylag(N+1:2*N)+y(1:N)+y(2*N+1));
dydt(2*N+1:3*N) = r2*(mat*y(2*N+1:3*N)+up3+down3)
+y(2*N+1:3*N).*(1-ylag(2*N+1:3*N)-y(N+1:2*N));
dydt=dydt';
end
end
```

A szimuláció kimenete:



51. Ábra – Zsákmány-ragadozó rendszer diffúzióval kimenete

3. DISZKRÉT RENDSZEREK (DTSS)

A diszkrét rendszerek megszámolható mennyiségű állapottal rendelkeznek, amik diszkrét időpillanatokban változhatnak. Adott időpillanatban a modell egy adott állapotban van, és meghatározza, hogyan változik ez az állapot. A következő állapot az aktuális állapottól és az állapotátmeneti függvénytől függ. A diszkrét rendszerek alkategóriái specifikáció függvényében a diszkrét idejű rendszerek (DTSS) és a diszkrét eseményű rendszerek (DEVS). A diszkrét idejű modellek általában a dinamikus modellek valamennyi formája közül a legintuitívabbak, ezekkel a következő, 3.1 alfejezet foglalkozik. A DTSS segítségével megadhatók matematikai számsorozatok, illetve populációs modellek bizonyos típusai, valamint a diszkrét idejű térbeli rendszerek a sejtautomaták specifikációjával foglalkozik.

A diszkrét idejű rendszereknek számos alkalmazása van. A legnépszerűbbek a digitális rendszerekben, ahol az óra határozza meg a diszkrét időlépéseket, de a diszkrét idejű rendszereket gyakran használják folytonos rendszerek közelítőjeként is. Itt egy időegységet választanak, pl. egy másodpercet, egy percet vagy egy évet, hogy meghatározzák a mesterséges órát, és a rendszert úgy ábrázolják, ahogyan az állapot változik az egyik „megfigyelési" pillanatról a másikra. Ezért egy diszkrét idejű modell felépítéséhez meg kell határoznunk, hogy az aktuális állapot és a környezetből érkező bemenet, hogyan határozza meg a modell következő állapotát.

A diszkrét időmodellekben az idő diszkrét lépésekben halad előre, amelyekről feltételezzük, hogy valamilyen alapidőszak, például 1 másodperc, 1 nap vagy 1 év egész számú többszörösei.

Ha az állapot t időpontban s, és a bemenet t időpontban x, akkor a $t + 1$ időpont állapota $\delta\ (s, x)$ lesz, és a t időpont kimenete $\lambda\ (s, x)$ lesz.

Itt δ az állapotátmeneti függvény, és a táblázat első három oszlopának absztraktabb fogalma. λ a kimeneti függvény és az első két és az utolsó oszlopnak felel meg. A λ és s absztraktabb formák az átmeneti és kimeneti információk megadásának általánosabb módját jelentik. Ezeket tömörebben a következőképpen foglalhatjuk össze:

$$\begin{aligned} \delta(s, x) &= x \\ \lambda(s, x) &= x \end{aligned} \tag{3.1}$$

amelyek szerint a következő állapotot és az aktuális kimenetet egyaránt az aktuális bemenet adja. A λ és δ függvények is sokkal általánosabbak, mint a táblázat. Akkor is elgondolhatók és leírhatók, ha az állapotok vagy bemenetek minden kombinációjára fárasztó lenne táblázatot írni, illetve akkor is, ha nem végesek, és így táblázat írása egyáltalán nem lehetséges. Az állapotok *s(0), s(1), s(2),...* sorozatát állapotpályának nevezzük. Egy tetszőleges *s(0)* kezdeti állapotot véve, a sorozat további állapotait a következők határozzák meg:

$$s(t + 1) = \delta(s(t), x(t)) \tag{3.2}$$

Hasonlóképpen, a megfelelő kimeneti pálya a következőképpen adódik

$$y(t) = \lambda(s(t), x(t)) \tag{3.3}$$

3.1. Diszkrét idejű rendszerek (DTSS)

A DTSS *(Discrete Time System Specification)* diszkrét idejű rendszerek olyan rendszerek, amelyeket differencia egyenletek segítségével írnak le. A diszkrét jel impulzusok sorozata. A diszkrét idejű modellezés során, az állapotátmeneti függvény, az aktuális állapot alapján adott információt, a következő időlépésben, diszkrét formában és nem folytonosan írja le. Teszi ezt azért – ahogyan arra már a fejezet bevezetésében is utaltunk –, mert a modell a változóit, a természetes számok halmazán értelmezi. A diszkrét idejű rendszerek tehát, olyan diszkrét rendszerek, amelyek csak bizonyos (meghatározott) időbeli lépésekben változnak. Az ilyen rendszerekben a folyamatos jelekből diszkrét jeleket kapunk, mintavételi eljárással. Ezen rendszerek egyik gyakori felhasználási módja az automatákban mutatkozik meg, melyekről későbbi fejezeteinkben még részletesebb leírást adunk. Formálisan ez a rendszer felírható, mint

$$DTSS = (X, Y, S, \delta, \lambda, c)$$

ahol:

- X – bemenetek halmaza
- Y – kimenetek halmaza
- S – állapotok halmaza
- $\delta: S \times X \rightarrow S$ – az állapotátmenet függvény
- $\lambda: S \rightarrow Y$ (Moore-típus)
- c – az időpillanatot meghatározó konstans

Ez a modell periodikus ütemben ellenőrzi bemeneteit, és az állapotinformáció alapján kimenetet állít elő, valamint megváltoztatja belső állapotát.

3.1.1. Matematikai számsorozatok

A következőkben tekintsünk meg néhány olyan matematikai számsorozatot, melyekkel már középiskolában is találkozhattak a diákok.

3.1.a. Példa – Fibonacci számsorozat

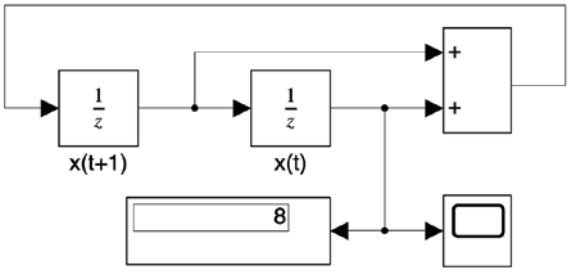
A sorozat Fibonacci vagy másnéven Leonardo di Pisa olasz matematikusról kapta a nevét, aki a számsor megalkotásakor az alábbi problémából indult ki:

Hogyan változik a nyulak populációja egy kezdeti pár esetén, ha tudjuk, a nyulak két hónap alatt válnak ivaréretté, és ezután minden pár, minden hónapban egy új párnak ad életet, és mindegyikük életben marad?

A feladat megoldásában a nyúl-párok számának időbeli alakulását kell követni. Az első hónapban egy nyúl-párunk van, és ugyanannyi lesz a másodikban is. A párok száma csak a harmadik hónapban változik egyről kettőre. A következő hónapban a szülők újabb párnak adnak életet, így a párok száma háromra nő. Az ötödik hónapban azonban már az új pár is szaporulatképes, így az új párok száma kettővel nő, és az összes párok száma ötre gyarapodik. A következő hónapban már mindkét ifjabb generáció hoz létre utódokat, és a párok száma hárommal növekedve nyolcra változik. Ezt a folyamatot folytathatnánk egész addig, amíg el nem jutunk abba a t időbe, ameddig megszeretnénk határozni a sorozat hosszát. A sorozat előállításának alapja az a tulajdonság, hogy a harmadik elemtől kezdve bármely elem az előző kettő összege (Csikó, 2015).

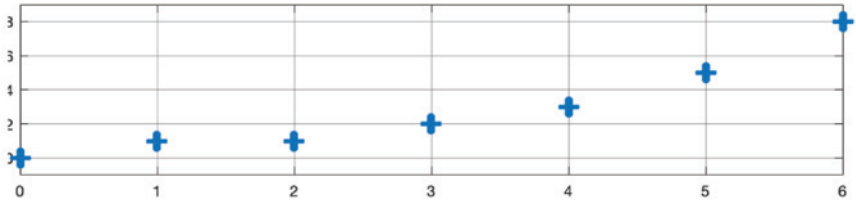
$$x(t) = x(t-1) + x(t-2)$$
$$x(t+2) = x(t+1) + x(t) \qquad (3.4)$$

Simulink modell:



52. Ábra – Fibonacci számsor kiszámítása Simulink-ben

Szimuláció kimenete:



53. Ábra – Fibonacci számsor

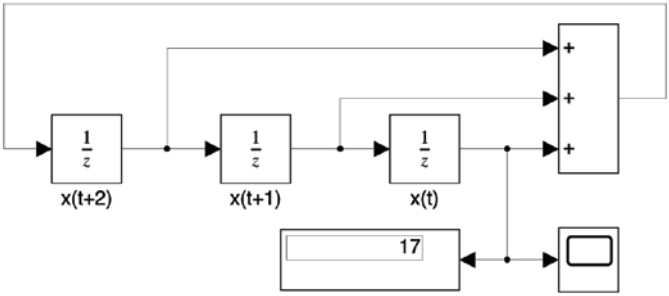
Leolvasott értékek: 0, 1, 1, 2, 3, 5, 8

3.1.b. Példa - Rekurzív számsorozat

Az alábbi is, egy rekurzív módon felírt számsor:

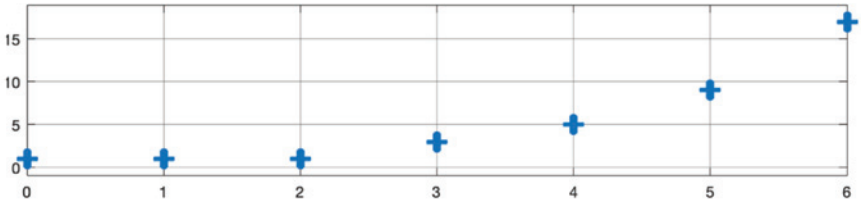
$$x(t+3) = x(t+2) + x(t+1) + x(t) \tag{3.5}$$

Simulink modell:



54. Ábra – Rekurzív számsor kiszámítása Simulink-ben.

Szimuláció kimenete:

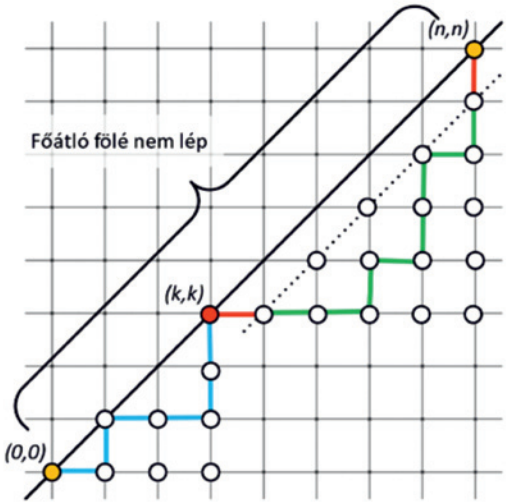


55. Ábra – Rekurzív számsor

Leolvasott értékek: 1, 1, 1, 3, 5, 9, 17

3.1.c. Példa - Catalan számsorozat

Ezen rekurzív számsorozatot egy egyszerű példával ismertetjük. Hányféleképpen lehet eljutni a $Z \times Z$ négyzetrácson a $(0, 0)$ pontból az (n, n) pontba (egységhosszú) $\rightarrow$ és $\uparrow$ lépésekkel úgy, hogy soha nem megyünk az $y = x$ egyenes fölé?



56. Ábra – Példa a lépési kombinációk lehetőségeire

Legyen $n \geq 1$. Ekkor minden megszámolandó útnak a végpontot leszámítva is van pontja az $y = x$ átlón. A

$$C_0 = 1,$$

$$C_n = \sum_{k=0}^{n-1} C_k \, C_{n-1-k}, \qquad \text{ha } n \geq 1 \tag{3.6}$$

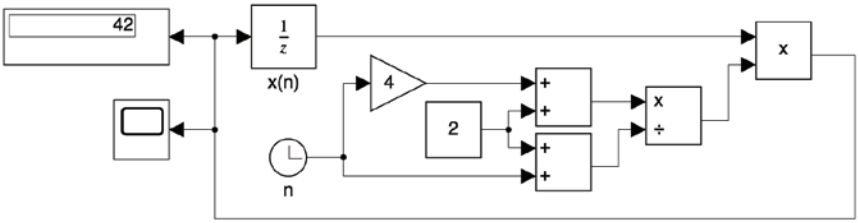
rekurzióval kigenerált sorozat elemeit Catalan számsornak nevezzük, miszerint:

$$C_0 = 1$$
$$C_1 = C_0 \, C_0 = 1$$
$$C_2 = C_0 \, C_1 + C_1 \, C_0 = 2$$
$$C_3 = C_0 \, C_2 + C_1 \, C_1 + C_2 \, C_0 = 5$$
$$\vdots$$

ahol C_k C_{n-1-k} azokat az utakat számolja meg, amelyek a *(k, k)* pontban lépnek utoljára az átlóra a végpont előtt *(k = 0, 1, ..., n−1)*. Ebből adódóan elmondható, hogy a Catalan-szám megszámolja azokat a → és ↑ lépésekből álló (0,0) ⇝ *(n, n)* utakat, amelyek soha nem lépnek az *y = x* egyenes fölé (Nagy, 2016). A rekurzív számsorunkat az alábbi módon modellezhetjük:

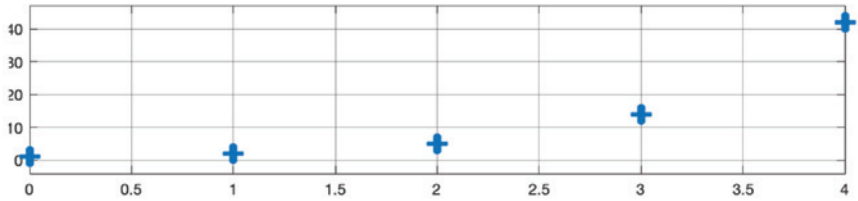
$$x(n+1) = \frac{2(2n+1)}{n+2} + x(n), \qquad ahol\ x(n) = 0$$
$$x(n+1) = \frac{4n+2}{n+2} + x(n) \tag{3.7}$$

Simulink modell:



57. Ábra – Catalan számsor kiszámítása Simulink-ben

Szimuláció kimenete:



58. Ábra – Catalan számsor

Leolvasott értékek: 1, 1, 2, 5, 14, 42

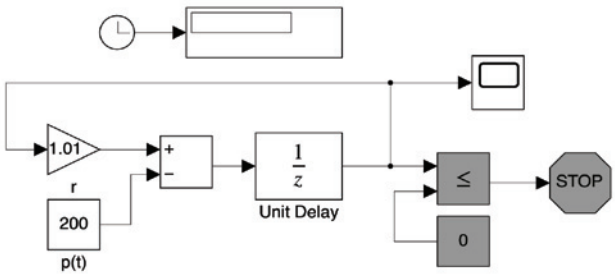
3.1.2. Kölcsöntörlesztés

A kölcsöntörlesztés egy tipikus diszkrét idejű modell. Az alapfelvetés, hogy a bank egy bizonyos nagyságú kölcsönt ad a kliensnek, akinek havi rendszerességgel törlesztenie kell, figyelembe véve a havi kamatláb mértékét. Mivel a törlesztés – és ezáltal a változás – általában havi egy alkalommal történik, így felesleges folytonos időben vizsgálni a modellt. A modellünk kifejezhető, mint:

$$b(t) = rb(t-1) - p(t)$$
$$r = i + 1 \tag{3.8}$$

ahol:
- $b(t)$ – Kölcsön összege t időpontban
- $p(t)$ – Havi törlesztés összege
- i – Havi kamatláb

Simulink modell:

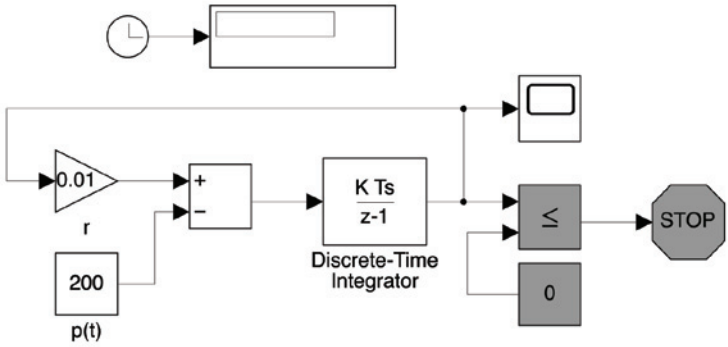


59. Ábra – Kölcsöntörlesztés modell megvalósítása Simulink-ben

Diszkrét integrál segítségével

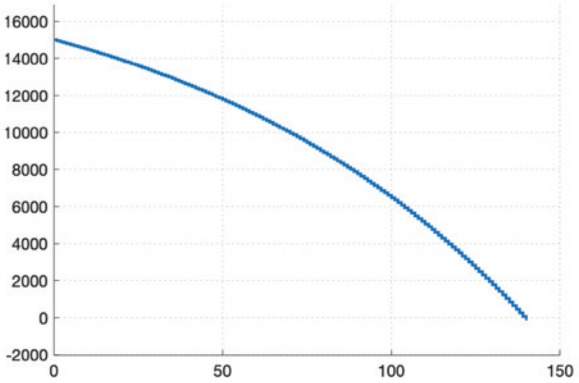
$$b(t) = b(t-1) + \int_{\Delta(t-1)}^{\Delta t} (ib(u-1) - p(u))du \tag{3.9}$$

Simulink modell:



60. Ábra – Kölcsöntörlesztés modell diszkrét integrállal megvalósítása Simulink-ben

Szimuláció kimenete $p = 200€$, $i = 0.01$ és $b(0) = 15000€$ esetén a törlesztési idő kb. 140 hónap, közel 12 év:



61. Ábra – Kölcsöntörlesztés vizualizációja

3.1.3. Populációs modellek

Ahogy folytonos esetben láthattuk, az olyan egyszerű népesedési modellek, mint a Malthus-féle (exponenciális) és a Verhulst-féle (logisztikus) modellek természetes kiindulópontot jelentenek a demográfiai folyamatok tanulmányozásában. A következőkben tekintsük meg ezen modellek diszkrét idejű eseteit (Brauer & Castillo-Chavez, 2012).

3.1.d. Példa - Exponenciális növekedési modell

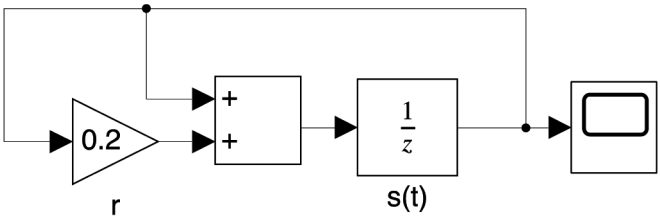
Exponenciális növekedés modellje alatt egy olyan egyszereplős nyitott rendszert értünk, ahol a nagy méretű társulásban általában több utód is születik, így a populációban szereplő egyedek hajlamosak a túlszaporodásra. Legyen

$$s(t+1) = s(t) + rs(t) \tag{3.10}$$

ahol:

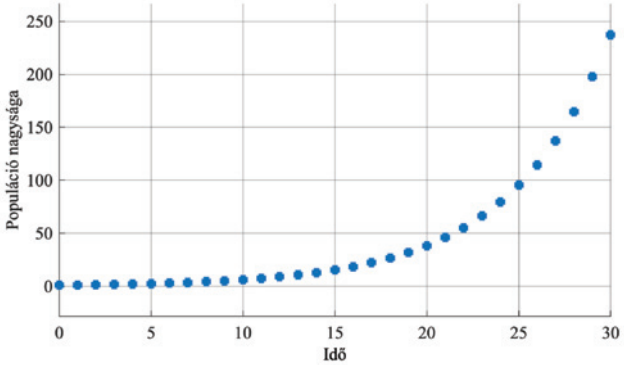
- $s(t)$ – Populáció nagysága
- r – Növekedési ráta
  - $r = b - d$
  - b – Születések száma
  - d – Halálozások száma

Simulink modell:



62. Ábra – Exponenciális növekedési modell (diszkrét időben) megvalósítása Simulink-ben

Szimuláció kimenete $r = 0.2$ és $s(0) = 1$ esetén:



63. Ábra – Exponenciális növekedési modell (diszkrét időben) szimulációja

3.1.e. Példa - Logisztikus növekedés modellje

Az exponenciális növekedés modelljét továbbfejlesztve beszélhetünk, a logisztikus növekedés modelljéről. Ebben az esetben egy zárt populáción belül, véges sok erőforrás érhető el. Ezt a következőkben nevezzük kapacitásnak. A modell leírja az ezen erőforrásokért folytatott küzdelmet. Az adott körülmények (zárt populáció, véges sok erőforrás) kihatással lesznek a társulás méretére, így egy bizonyos csökkenés figyelhető meg, ha a kezdeti állapot meghaladja a kapacitás értékét. Abban az esetben, ha a populáció kezdeti értéke kisebb, mint a kapacitás, és pozitív növekedési rátával számolunk, akkor végkimenetként *S* alakú görbét kapunk, ahol a grafikonunk tetején közelít a populáció a legjobban a maximális kapacitás értékéhez. A modell kifejezhető, mint:

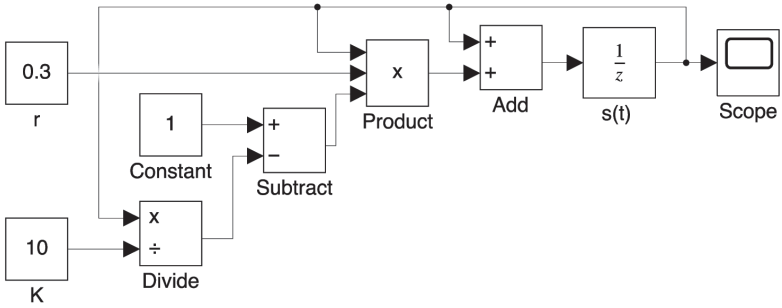
$$s(t+1) = s(t) + rs(t)\left(1 - \frac{s(t)}{K}\right)$$

$$s(t+1) = s(t) + rs(t) - \frac{rs(t)s(t)}{K} \tag{3.11}$$

ahol:

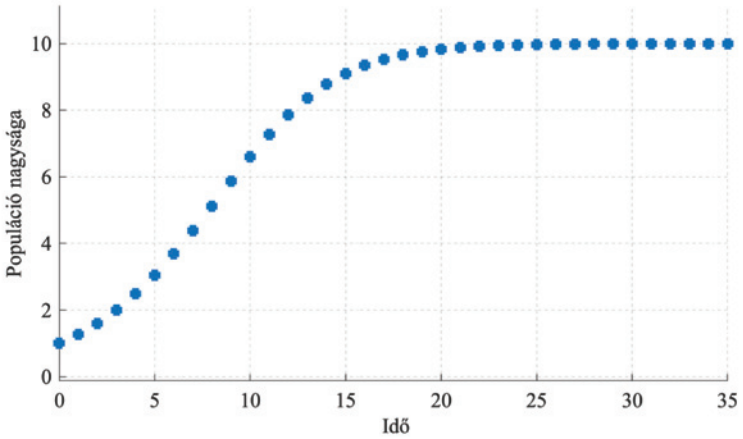
- $s(t)$ – Populáció nagysága
- K – Kapacitás
- r – Növekedési ráta
  - $r = b - d$
  - b – Születések száma
  - d – Halálozások száma

Simulink modell:



64. Ábra – Logisztikus növekedési modell (diszkrét időben) megvalósítása Simulink-ben

Szimuláció kimenete $r = 0.3, K = 10, S(0) = 1$ esetén:



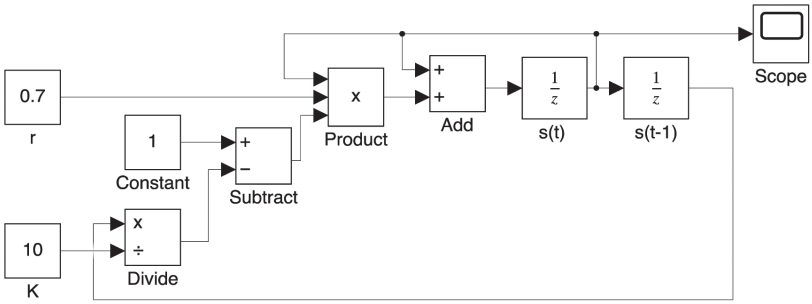
65. Ábra – Logisztikus növekedési modell szimulációja

3.1.f. Példa - Logisztikus növekedés modellje időeltolással

$$s(t+1) = s(t) + rs(t)\left(1 - \frac{s(t-1)}{K}\right) \tag{3.12}$$

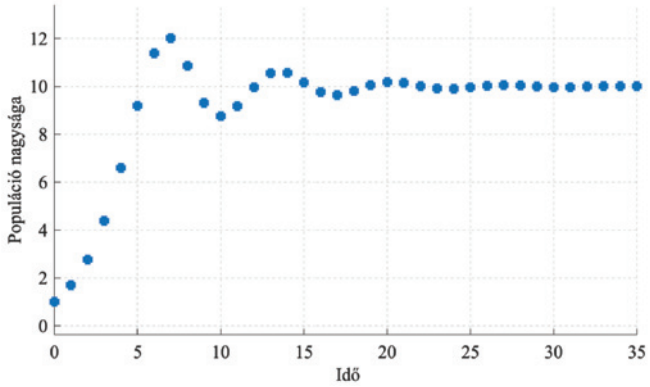
$$s(t+1) = s(t) + rs(t) - \frac{rs(t)s(t-1)}{K}$$

Simulink modell:



66. Ábra – Időeltolásos logisztikus növekedési modell (diszkrét időben) megvalósítása Simulink-ben

Szimuláció kimenete $r = 0.7, K = 10, s(0) = 1$ esetén:



67. Ábra – Időeltolásos logisztikus növekedési modell szimulációja

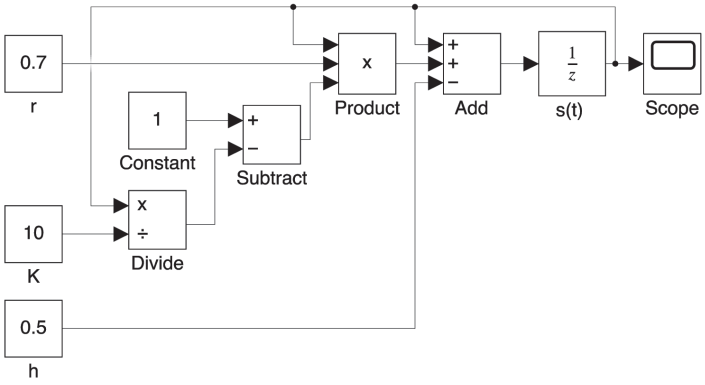
3.1.g. Példa - Horgászati modell

$$s(t+1) = s(t) + rs(t)\left(1 - \frac{s(t)}{K}\right) - h \tag{3.13}$$

ahol:

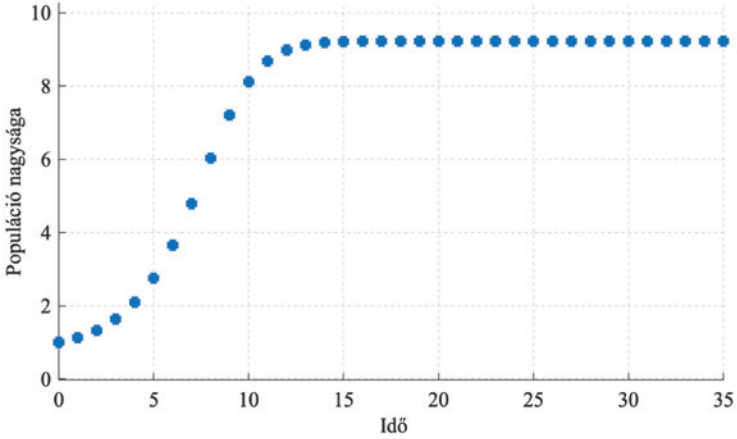
- $s(t)$ – Populáció nagysága
- K – Kapacitás
- h – Regulációs tényező
- r – Növekedési ráta
  - $r = b - d$
  - b – Születések száma
  - d – Halálozások száma

Simulink modell:



68. Ábra – Horgászati modell megvalósítása Simulink-ben

Szimuláció kimenete $r = 0.7, K = 10$, $h = 0.5$ és $s(0) = 1$ esetén:



69. Ábra – Horgászati modell szimulációja

3.1.h. Példa – Zsákmány-ragadozó modell

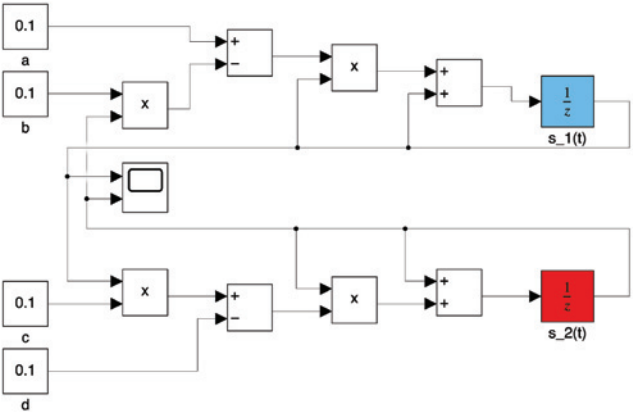
Ebben az esetben két populáció egymásra gyakorolt hatását vizsgáljuk meg (pl. rókák és nyulak kapcsolatát). A modell sajátossága, hogy az egyik egyed populációjának a nagysága a másik faj tápanyagmennyiségét jelöli. Az egyik faj egyedszámának a változása tehát kihatással van a másikéra. Amennyiben a nyulak elszaporodnak, az a rókák számának a növekedését is jelenti, azonban, ha a zsákmány egyedszáma lecsökken, akkor a ragadozók száma is redukálódni fog. Ez kifejezhető, mint:

$$\begin{aligned} s_1(t+1) &= s_1(t) + s_1(t)(a - bs_2(t)) \\ s_2(t+1) &= s_2(t) + s_2(t)(cs_1(t) - d)) \end{aligned} \tag{3.14}$$

ahol:

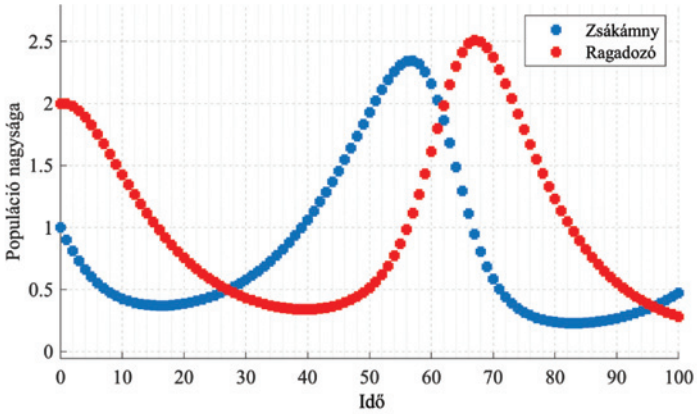
- s_1 – a zsákmány egyedszáma
- s_2 – a ragadozók egyedszáma
- b – a ragadozó populáció zsákmányra gyakorolt hatása
- d – a ragadozók halálozási rátája

Simulink modell:



70. Ábra – Predáció modell megvalósítása Simulink-ben

Szimuláció kimenete:



71. Ábra – Predáció modell szimulációja

3.1.i. Példa - Feigenbaum – bifurkációs diagram

A matematikában, különösen a dinamikus rendszerekben, a bifurkációs diagram egy rendszer aszimptotikusan megközelített értékeit – fixpontokat – mutatja, a rendszer egy bifurkációs paraméterének függvényében.

A logisztikus leképezést a következő iterációs lépés definiálja:

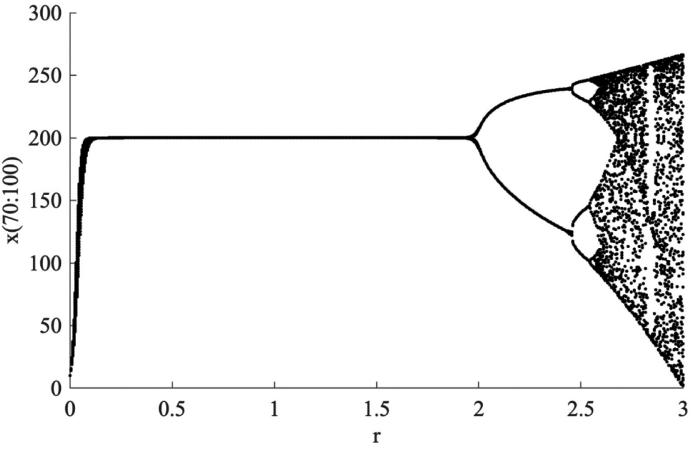
$$s_{n+1} = rs_n (1 - s_n)$$

ahol r az úgynevezett kontroll vagy bifurkációs paraméter (Rudolf, 2012). Az r függvényében a leképezés kvalitatíve különböző módokon viselkedhet. Lehet benne egy, vagy több fixpont. A bifurkációs diagram a stabil pályák periódusainak elágazását mutatja 1-től 2-ig, 4-ig, 8-ig stb. E bifurkációs pontok mindegyike periódusduplázódási bifurkáció. Az egymást követő r értékek közötti intervallumok hosszának aránya, amelyeknél a bifurkáció bekövetkezik, az első Feigenbaum-állandóhoz konvergál. A Feigenbaum-állandó két matematikai állandó, amelyek arányokat fejeznek ki bifurkációs diagramban egy nemlineáris leképezésben (lásd 71. Ábra).

14. Forráskód – Bifurkációs diagram

```
clear all
close all

K = 200;
hold on
for r=0:0.005:3
  x(1) = 10;
  for i = 2:100
    x(i) = x(i-1)+r*x(i-1)*(1-x(i-1)/K);
  end
plot(r,x(1,70:100),".k")
end

xlabel r
ylabel x(70:100)
hold off
```

Kimenet:



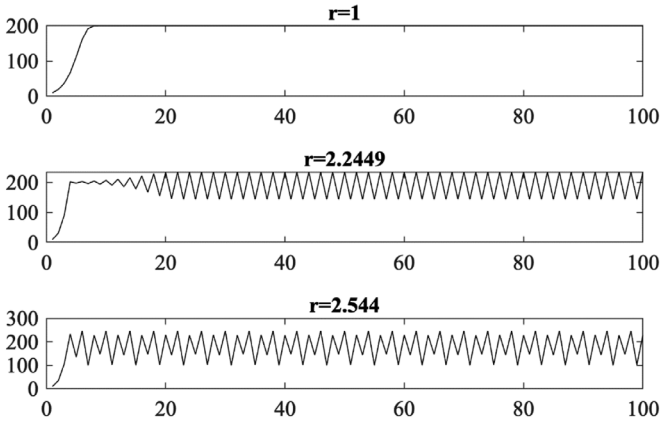
72. Ábra – Feigenbaum diagram

15. Forráskód – Feigenbaum diagram

```
clear all
close all

K = 200;
r = 1;
x1(1) = 10;
for i = 2:100
  x1(i) = x1(i-1)+r*x1(i-1)*(1-x1(i-1)/K);
end
r = 2.2449;
  x2(1) = 10;
for i = 2:100
x2(i) = x2(i-1)+r*x2(i-1)*(1-x2(i-1)/K);
end
r = 2.544;
  x3(1) = 10;
for i = 2:100
x3(i) = x3(i-1)+r*x3(i-1)*(1-x3(i-1)/K);
end
```

```
20
21        hold on
22        tiledlayout(3,1)
23        nexttile
24        plot(x1, "*k"); plot(x1, "k"); title r=1
25        nexttile
26        plot(x2, "*k"); plot(x2, "k"); title r=2.2449
27        nexttile
28        plot(x3, "*k"); plot(x3, "k"); title r=2.544
29        hold off
```

Kimenet:



73. Ábra – Szimuláció kimenetei különböző növekedési ráták esetén

3.2. Diszkrét idejű térbeli rendszerek (CA)

A matematikus Stephen Wolfram definíciója alapján, a sejtautomaták természetes rendszerek egyszerű matematikai idealizációi. Ezek rácsból és egyforma diszkrét mezőkből állnak. Minden mező véges számú integer értéket képes felvenni. A mezők értékei diszkrét időbeli lépésenként fejlődnek a determinisztikus szabályoknak megfelelően, amik meghatározzák minden egyes mező értékét a szomszédos mezők értékének függvényében. Ennek értelmében a sejtautomaták, természetes rendszerek leírására gyakran használt, parciális differenciálegyenletek diszkrét idealizációjának tekinthetők.

A sejtautomaták olyan diszkrét, nemlineáris dinamikus rendszerek, amelyek azonos típusú „sejtekből” állnak. Viselkedésüket teljes mértékben meghatározza jelenlegi saját állapotuk és a közvetlen szomszédságukban lévő „sejtek” állapotai. Az, hogy mit tekintünk környezetnek (szomszédságnak), az az adott szomszédság definíciójától függ. A rendszer ideje, tere és állapota diszkrét. A sejteknek általában nincs memóriájuk, és egy szabályos rácson vannak elosztva (egy-, két vagy több dimenziósok, lehetnek négyzet-, háromszög- vagy hatszögletű rácsok).

Ezeknek a térben és időben diszkrét dinamikus rendszereknek fő jellemzői közé tartozik, a ráccsal és mezőkkel kifejezett rendszeres sejtstruktúra, ami N-dimenziós (leggyakrabban 1D és 2D) térben helyezkedik el. A szélső értékek érdekében érdemes megemlíteni, hogy mivel nincs egyik oldalon szomszédjuk, gondoskodni kell a kezelésükről. Ez megoldható úgy, hogy az adott generáció két szélén található elemet szomszédosnak vesszük. Ebben az esetben a határ ciklus, vagy toroid formát vesz fel. Az adott rendszerben szereplő sejtek mindegyikére igaz, hogy a véges számú lehetséges állapot egyikét megszerezhetik. A sejtautomaták működését helyi kölcsönhatások írják le, a sejtek „látása” erősen korlátozott, minden sejt csak a közvetlen környezetét ismeri. A diszkrét időbeli lépések lehetővé teszik, hogy minden egyes lépésben, minden egyes cella értéke egyidejűleg frissüljön. A lépések során a cella állapota változhat, és ez meghatározható az adott cella és környezete aktuális állapota, valamint a rávonatkozó szabály (vagy függvény) alapján. Fontos megjegyezni, hogy az összes cellára azonos interakciós szabályok vonatkoznak, illetve azt sem szabad figyelmen kívül hagyni, hogy az összes cella állapotát a vele szomszédos cellák állapotai befolyásolják.

Felhasználásukat tekintve ezek a modellek főleg biológiai, fizikai és társadalmi jellegű folyamatok szimulációjára alkalmasak, mint például:

- forgalom szimuláció,
- ipari műveletek és folyamatok modellezése,
- erdőtüzek és más természeti katasztrófák terjedésének és hatásainak szimulációja,
- járványok terjedésének szimulációja és előrejelzése.

Alapfogalmak

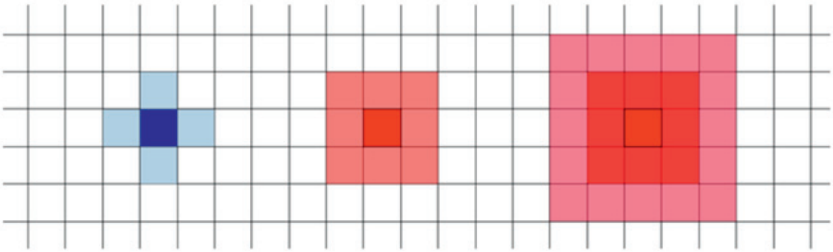
Sejttér, élettér – Az adott sejtautomatában meghatározott terület, elemi sejtek hálózata. A tér elméletileg végtelen, szabályos és tetszőlegesen meghatározott dimenzióban valósul meg.

Cella, sejt – A rendszer alapegysége, a terület rács általi egyenlő részekre való bontása. Véges számú diszkrét állapotuk és átmeneti szabályuk van.

Állapot – A rendszer minden sejtje rendelkezik állapottal, amit a közvetlen környezete és a sejtekre vonatkozó szabályok határoznak meg. A sejtek állapotát minden generációban párhuzamosan számoljuk a helyi átmeneti függvény alapján.

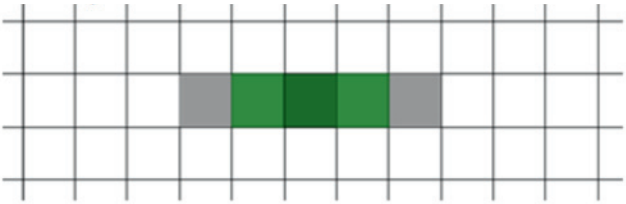
Szomszédság – A élettér és rács függvényében minden sejtnek vannak szomszédjai, például a (gyakran használt) kétdimenziós négyzetháló esetén maximum 8 szomszédja lehet egy sejtnek. Szomszédságból megkülönböztetünk Neumann és Moore féle szomszédságot. Előbbi esetében csak a közös rácsél jelent szomszédságot (74. Ábra, kék sejt), míg utóbbinál közös él és közös csúcs is szomszédságként van meghatározva (74. Ábra, piros sejt). Adott sejt közvetlen környezete befolyásolja a sejt jövőbeli állapotát. Egy dimenzió esetén a környezetet a sugár *(r)* határozza meg. A 74. Ábrán, a legismertebb 2D sejtautomaták szomszédságai láthatók. Egydimenziós és hatszögletű rácsok esetén, csak élek szerint lehet szomszédságot vizsgálni.

Szabályok – Függvények határozzák meg az egyes cellák állapotváltozását, és mindig egyszerre számolódnak. Ezek változói a vizsgált sejt és környező cellák állapotainak aktuális értékei. A szabályokat, amelyek megadhatjuk szöveges, vagy grafikus formában (Kmeť, Modellezés és szimuláció elmélete és eszközei, 2021).



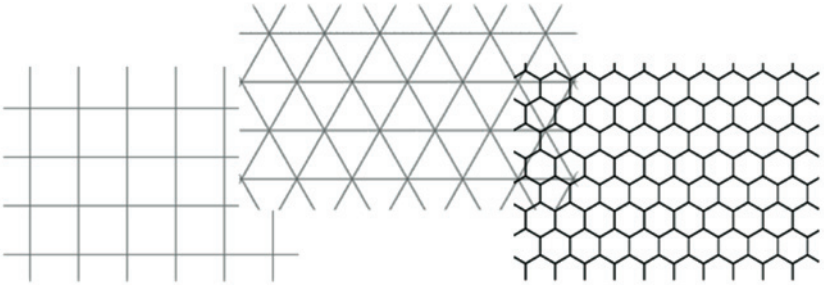
74. Ábra – Neumann szomszédság (kék) és Moore szomszédság r=1 és r=2 esetén

Rács típusai
1D – központi cella és környezete



75. Ábra – 1D cella (zöld) és szomszédsága r = 1 (világos zöld) és r = 2 (szürke) esetén

2D – leggyakrabban használt rácstípusok
- Négyzetrács
- Háromszögű rács
- Hatszögű rács



76. Ábra – Négyzetrács, három és hatszögű rács

A mobil automata a számítástudomány elméletén belül egy olyan automata osztály, amely hasonló a sejtautomatákhoz, de itt egyetlen „aktív" cella van, nem pedig az összes cella frissül párhuzamosan. A mobil automaták három fő jellemzővel rendelkeznek, amelyek nagyban meghatározzák, hogy milyen szimulációkra alkalmas az adott modelltípus.

Párhuzamosság – Minden elem (sejt) állapota új értékének kiszámítása egyszerre történik.
Elhelyezkedés – Egy elem új állapota csak az eredeti állapotától és a környezete elemeinek eredeti állapotától függ.
Homogenitás – Az automata minden egyes sejtjére ugyanaz az átmeneti függvény vonatkozik.

Sejtautomaták meghatározása

$$CA = (B, S, O, f)$$

ahol:

- CA – Sejtautomata (Cellular Automata)
- B – véges cellakészlet
- S – véges állapothalmaz
- O – környezet
- f – átmeneti függvény*t*

$$s(t + 1) = f\,(s(t), O_1(t), O_2(t), O_3(t), \ldots)$$

3.2.1. Egyszerű sejtautomaták

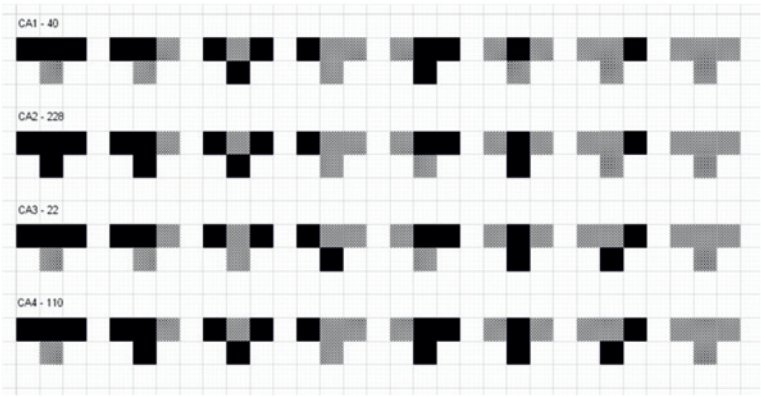
A következőkben a legegyszerűbb egydimenziós sejtautomatákra hozunk példákat.

3.2.a. Példa - Wolfram sejtautomatái

Stephen Wolfram (1959 -) informatikus, fizikus és a Wolfram Research vezérigazgatója. A sejtautomaták kutatásának új lendületet adott az 1D automaták tulajdonságainak tanulmányozásával. Az egydimenziós sejtautomaták, a legegyszerűbb osztályba tartoznak. Ezen elemi sajtautomaták minden egyes sejtje két lehetséges érték valamelyikével (0,1), és szabályokkal rendelkezik. A szabályok kizárólag a közvetlenül szomszédos cellák értékeitől függnek. Ennek eredményeképp, adott sejtautomata teljesen leírható a következő generáció képzését bemutató táblázattal. Mivel minden sejtnek két értéke lehet, és egydimenziós automata esetén adott sejtnek jobb és bal szomszédja van, ez $2 \cdot 2 \cdot 2 = 2^3 = 8$ különböző állapotkombinációt jelent, amik bináris (0,1) értékek esetén $2^8 = 256$ különböző elemi sejtautomata meghatározását teszik lehetővé. A szabályok könnyedén felírhatóak bináris alakban, például a 30. szabály bináris alakja 00011110. Az egydimenziós sejtautomaták könnyen ábrázolhatók kétdimenziós térben, ahol minden egyes sor egy új generációt jelent. Gyakran az első sor középső értékét határozzák meg „élőként" és az adott automata szabálykészlete alapján leírják a további generációkat. A különböző szabályok, különböző mintákat hoznak létre, amelyek lehetnek egyszerűek, ismétlődőek, kaotikusak, de akár fraktálokhoz hasonló tulajdonsággal is rendelkezhetnek.

Wolfram a 256 sejtautomatát, bonyolultságuk szerint, 4 kategóriára osztotta fel. Ez az osztályozás a többdimenziós sejtautomatáknál is használatos. A kategóriák nevei CA1, CA2, CA3, CA4.

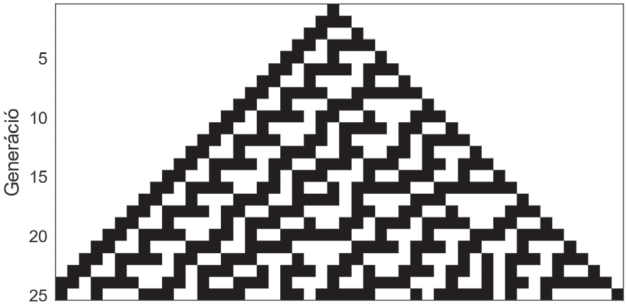
- CA1 – Az egész sort kiürítik, vagy kitöltik, pl. 40. szabály
- CA2 – A kezdeti minták és aktivitás helyett fokozatosan ciklikus, ismétlődő struktúrák veszik át vezetést, pl. 109. és 228. szabály
- CA3 – Szabad szemmel nem ismerhető fel benne minta, zajszerű, a minta fejlődése kaotikus, pl. 22. szabály
- CA4 – Összetett, de látható szabályszerűségeket mutató minták, pl. 110. szabály.



77. Ábra – Nevezetesebb szabályok minden kategóriából

Ha az automatában levő állapotok száma $m > 2$, akkor több értékkel dolgozunk. A lehetséges állapotok egy részét nyugalmi állapotnak nevezzük. Nyugalmi állapotnak nevezzük, ha a vizsgált cella értéke stagnál, és nem változik a következő generációra. Ha a nyugalmi cella szomszédai is nyugalmi cellák, akkor értéke nem fog változni a kövtkező iteráció során.

Bemenet	**111**	**110**	**101**	**100**	**011**	**010**	**001**	**000**
Szabály 0	0	0	0	0	0	0	0	0
…	…	…	…	…	…	…	…	…
Szabály 4	0	0	0	0	0	1	0	0
…	…	…	…	…	…	…	…	…
Szabály 8	0	0	0	0	1	0	0	0
…	…	…	…	…	…	…	…	…
Szabály 90	0	1	0	1	1	0	1	0
…	…	…	…	…	…	…	…	…
Szabály 180	1	1	1	1	1	1	1	1



78. Ábra – Wolfram 30. szabálya 25 iteráció után

Az 1D sejtautomaták környezetét sugár jellemzi, ez megegyezik a vizsgált szomszédok számaival. Wolfram elemi sejtautomatáinál $r = 1$ az értéke, de a választott rendszer igényei szerint eszközölhető változtatás (Weisstein, Elementary Cellular Automaton, 2022).

16. Forráskód – Wolfram celluláris automaták

```
clear all, close all, clc
%Paraméterek
rule = 30;

rule_bin = dec2bin(rule);
rule_array = zeros(1,8);
for i=1:length(rule_bin)
   rule_array(i+8-length(rule_bin)) = str2num(rule_bin(i));
end

input_array = zeros(3,8); % 1 - bal, 2-elem, 3-jobb

A = zeros(1,50);
A(25) = 1;

l = length(A);
for i = 2:25
   for j = 2:l-1
```

```
19      A(i,j) = inquiry(rule_array,A(i-1,j-1),A(i-
    1,j),A(i-1,j+1));
20
21      end
22      A(i,1) = inquiry(rule_array,A(i-1,1),A(i-
    1,j),A(i-1,j+1));
23      A(i,1) = inquiry(rule_array,A(i-1,j-1),A(i-
    1,j),A(i-1,1));
24  end
25
26  %Grafika
27  clims = [0 1];
28  colorbar
29  colormap(flipud(gray))
30  imagesc(A,clims)
31
32  axis equal
33  axis([1.5 49.5 0.5 25.5])
34  ylabel('Generáció')
35  xticklabels({„})
36  h=gca;
37  h.XAxis.TickLength = [0 0];
38  h.YAxis.TickLength = [0 0];
39  function new_state = inquiry(rule,i1,i2,i3)
40
41    i = bin2dec(strcat(num2str(i1),num2str(i2),num2str
    (i3)));
42    new_state = rule(8-i);
43  end
```

3.2.2. A 2D sejtautomaták

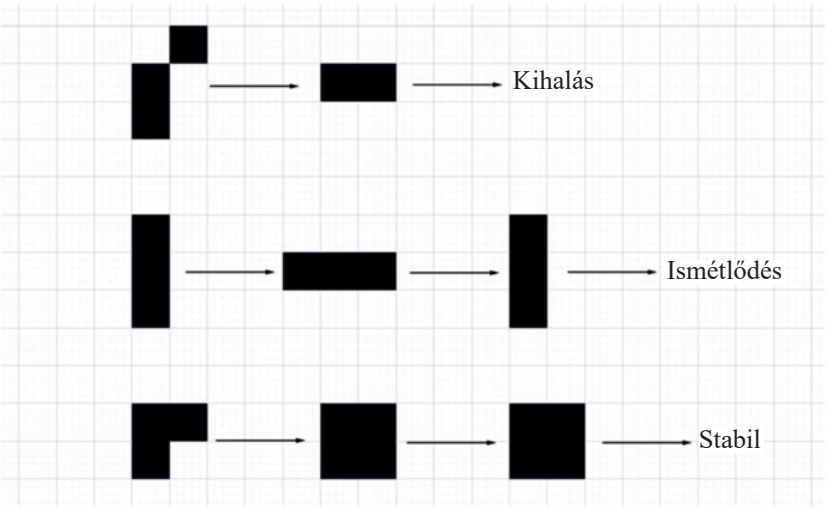
A 2D-s sejtautomaták esetében az élettér egy kétdimenziós terület. A fentebb tárgyalt sejtautomaták kirajzolásakor minden sor egy új generációt jelentett. A szomszédság vizsgálata ebben az esetben bonyolódik, hiszen négyzetrács esetén minden vizsgált szomszédság típusától függően 4, vagy 8 szomszédja lehet egyetlen cellának (74. Ábra). Két ismert képviselője a témakörnek Conway Game of Life modellje és Langton hangyája. Mindkét esetben igaz, hogy egyszerű, minimális szabályrendszer felel a modell működéséért (Packard & Wolfram, 1985).

3.2.b. Példa - Conway életjáték (Game of Life) modellje

John Horton Conway (1937-2020) angol matematikus. Kutatási területei közé sorolható többek között a számelmélet és kombinatorikus játékelmélet. Munkásságához kapcsolódik a szürreális számok meghatározása és az életjáték-modell (Game of Life, röviden LIFE).

Az életjáték-modell egy celluláris automata egyszerű felépítéssel és szabályrendszerrel. A játék területét kétdimenziós négyzetrács biztosítja, ahol bármely cella 8 darab másik cellával szomszédos. A modell Moore-féle szomszédságot vizsgál (tehát 8 darab szomszédos cellát) a felsorolt szabályokra. A rács által határolt celláknak két állapota lehetséges: élő és halott. Életüket az alábbi szabályok határozzák meg:

1. bármely kettőnél kevesebb szomszéddal rendelkező élő cella meghal elnéptelenedés következtében;
2. bármely 2, vagy 3 élő szomszéddal rendelkező cella életben marad a következő generációra;
3. bármely háromnál több élő szomszéddal rendelkező cella meghal túlnépesedés következtében;
4. bármely pontosan 3 élő szomszéddal rendelkező halott cella újra él, szaporodást szimbolizálva.

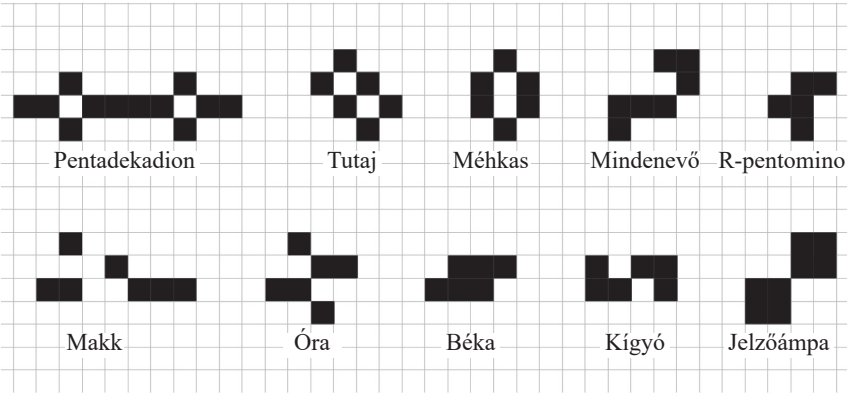


79. Ábra – Egyszerűbb alakzatok életciklusa

A játéktér területén lévő cellák vizsgálata egyidejűleg zajlik, és az adott kör végén, egyszerre lépnek életbe a változások. Az egyes iterációkat generációnak nevezzük. A szabályrendszer kialakításánál különös figyelmet kapott az egyszerűség, robbanásszerű növekedés lehetőségének elkerülése, illetve hogy kis kezdeti minták is képesek legyenek megjósolhatatlan, kaotikus kimeneteket generálni. A felsoroltak mellett potenciálisan lehetséges Von Neumann univerzális konstruktorok létrehozása, amelyek rendelkeznek az önreplikáció képességével.

A modellel való kísérletezés közben, kezdetben papíron, később számítógépen, nyomon követték egyes egyszerűbb alakzatok sorsát, és a jellegzetes alakzatokat névadás mellett 3 fő kategóriába sorolták:

- Tengődő alakzatok – statikusan stabil alakzatok, amelyek külső behatás nélkül nem változnak a generációk során. Ebbe a kategóriába tartoznak a cső, ló, kígyó, méhsejt és cipő alakzatok.
- Pulzáló alakzatok – dinamikusan stabil alakzatok, amelyek meghatározott számú generáció után újra felveszik a kezdő állapotukat. Ilyenek például a béka, óra és jelzőlámpa alakzatok.
- Űrhajók – az R-pentomino alakzatnál megfigyelték, hogy 1103 generáció szükséges a stabilizálásához, és közben távolodó sikló (glider) alakzatokat képez. A sikló alakzat érdekessége, hogy 4 generáció után éri el a kiindulási állapotát, és közben átlós mozgást végez a négyzethálón. A stabil irányba végzett mozgása miatt, ez lett az űrhajó kategória első tagja, és később a könnyű-, közép és nehézsúlyú űrhajók követték (Weisstein, Game of Life, 2022).



80. Ábra – LIFE modell néhány további ismert alakzata

17. Forráskód – Game of Life modell

```
clear all, close all, clc

A = zeros(50,50);
l = height(A);
h = length(A);
N = 20;
B = A;

X = [25,25,26,27,27,28,29,29,35,35,36,35];
Y = [25,26,25,26,28,29,29,28,33,34,34,35];

for i = 1:length(X)
  A(X(i),Y(i)) = 1;
end

plot(A)

for k = 1:N
  for i = 2:l-1
    for j = 2:h-1
      B(i,j) = A(i,j);
      count = calcNeighbor(A,i,j);
      if count > 3 || count < 2
        B(i,j) = 0;
      elseif A(i,j) == 0 && count == 3
        B(i,j) = 1;
      end
        end
      end

  A = B;
  plot(A)
end

```

```
35      function piece = calcNeighbor(A,x,y)
36        piece = 0;
37        for i = -1:1
38          for j = -1:1
39            piece = piece + A(x+i,j+y);
40          end
41      end
42        piece = piece - A(x,y);
43      end
44
45      function plot(A)
46        clims = [0 1];
47        colorbar
48        colormap gray
49        imagesc(A,clims)
50        pause(0.1)
51      end
```

A forráskód X és Y változóinak módosításával kipróbálható a felsorolt és tetszőleges alakzatok mozgása. Néhány példa:

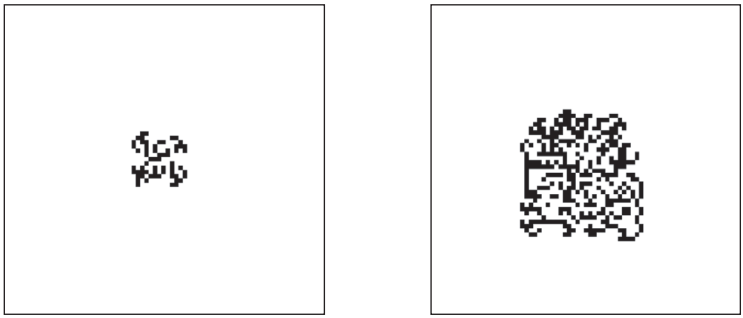
Kígyó: $X = [24, 25, 27, 24, 26, 27]$, $Y = [25, 25, 25, 26, 26, 26]$
Béka: $X = [24, 25, 26, 25, 26, 27]$, $Y = [25, 25, 25, 24, 24, 27]$
R-pentomino: $X = [25, 26, 24, 25, 25]$, $Y = [26, 26, 25, 25, 24]$

3.2.c. Példa - Langton hangyája

Langton hangyája egy népszerű Turing-gép, viszonylag egyszerű szabályrendszerrel. Chris Langton készítette a modellt 1986-ban, és csak egy fekete-fehér cellákból álló rácsra van szükség hozzá. A modell Turing-teljességét 2000-ben bizonyították, és azóta több különböző verziója készült el további állapotok és színek használatával. A modell érdekessége, hogy a hangya mozgása először véletlenszerűnek tűnik, viszont egy bizonyos mennyiségű lépés után végtelen ciklusba kerül. A négyzetrács néhány cellájában morzsa van, amik között bolyong a hangya. A hangya haladását befolyásolja, hogy milyen cellára lép előre. Ha fekete cellába kerül, akkor a cella színe fehérre változik és a hangya 90 fokkal balra fordul. Ha fehérre, akkor a cella színe feketére változik és jobbra tér. Ha a hangya morzsát talál, jobbra fordul, ha üres cellába érkezik, akkor tesz oda egy morzsát és balra fordul. Szimuláció indításakor az összes sejt fehér színűként van inicializálva.

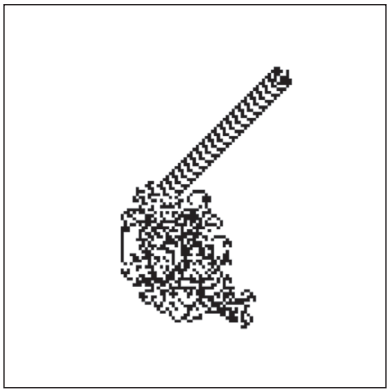
Az előbb említett egyszerű szabályok ellenére a modell meglepően összetett viselkedésre képes, a fő jellemzői a következők:

- Egyszerűség – Az első néhány száz lépés során a minták nagyon egyszerűek, gyakran szimmetriát mutatnak.
- Káosz – Pár száz lépés után a hangya által készített minta elkezd rendszertelenné válni és körülbelül a 10000. lépésig álvéletlennek tekinthető a viselkedése.



81. Ábra – Langton hangyája az egyszerűség és káosz fázisában (N = 500 és N = 5000)

- Rendeződés – 10000 lépés után, egy előzőktől teljesen eltérő jelenség kezdődik, a hangya belekezd egy önismétlő „főútvonalnak" (highway) nevezett minta építésébe, ami a végtelenségig ismétlődik.



82. Ábra – Langton hangyája rendeződés fázisában (N = 12000)

18. Forráskód – Langton hangyája

```
clear all, close all, clc

width = 80;
height = 80;
A = zeros(width,height);
N = 500;
pos = [width/2,height/2];
dir = 4;

for i=1:N
  ii = 0;
  jj = 0;
  if dir == 1
    ii = 1;
  elseif dir == 2
    jj = -1;
  elseif dir == 3
    ii = -1;
  else
    jj = 1;
  end

if A(pos(1)+ii,pos(2)+jj)==1
    dir = dir + 1;
    if dir == 5
      dir = 1;
    end
    A(pos(1)+ii,pos(2)+jj) = 0;
  else
    dir = dir - 1;
    if dir == 0
      dir = 4;
    end
    A(pos(1)+ii,pos(2)+jj) = 1;
  end
```

```
36          pos(1) = pos(1)+ii;
37          pos(2) = pos(2)+jj;
38        end
39
40        %Graphic
41        clims = [0 1];
42        colorbar
43        colormap(flipud(gray))
44        imagesc(A,clims)
45        axis equal
46        axis([0 width 0 height])
```

3.2.d. Példa - Silverman celluláris automatái

Brian Silverman nevéhez 3 ismertebb celluláris automata szabály kötődik (*Seeds* – Magok, *Brian's Brain* – Brian agya, *Wireworld* - Drótvilág). Ezek közül az előbbi kettő részletesebb bemutatása következik.[4]

Magok (Seeds) – Hasonlít Conway életjátékához, azonban néhány szabálybéli különbség miatt, jelentősen más a modell fejlődése. Az említett modell (17. Forráskód) a szabályrész módosításával alkalmas lesz az alábbi automaták (19. Forráskód, 20. Forráskód) futtatására is. A modell kétdimenziós végtelen élettérben valósul meg. A celláknak továbbra is két állapota van, élő és halott. A modell Moore-szomszédságot használ a szabály értelmezéséhez, amelyek a következőképp foglalhatók össze egy mondatban: A cella élővé válik, ha halott volt és pontosan 2 élő szomszéddal rendelkezett, minden más cella meghal. Azokat a mintákat, amelyek minden cellája meghal az adott generációban, főnixnek nevezzük. A szabály második feléből következő rövid életciklus ellenére az alacsony születési feltétel (2 élő szomszédos cella) miatt, a legtöbb minta, képes gyorsan terjeszkedni és vele párhuzamosan kaotikussá válni. Ebből kifolyólag Wolfram osztályozása szerint CA3 sejtautomata kategóriába esik. A program futtatásához elég tetszőleges kiindulási sejtmintát megadni, és az életjáték-modell (17. Forráskód) ciklusmagját módosítani az alábbi feltétel szerint (Wojtowitcz, 2006).

[4] https://conwaylife.com/wiki/Brian_Silverman (2024.08.02.)

19. Forráskód – Seeds

```
17        for k = 1:N
18          for i = 2:l-1
19            for j = 2:h-1
20              B(i,j) = A(i,j);
21              count = calcNeighbor(A,i,j);
22              if A(i,j) == 1
23                B(i,j) = 0;
24              elseif A(i,j) == 0 && count == 2
25                B(i,j) = 1;
26              end
27            end
28          end
29          A = B;
30
31          plot(A)
32        end
```

Brian agya *(Brian's Brain)* – A modell tekinthető a Seeds modell folytatásának, vagy továbbfejlesztésének. Szintén végtelen kétdimenziós életteret használ, azonban ebben az esetben a sejtek 3 állapottal rendelkeznek (élő, haldokló, halott). Az elérhető élő, haldokló és halott állapotok tükrében a szabályrendszer kiegészül azzal, hogy minden élő cella, haldokló állapotba kerül. Ez az állapot nem számít élőnek a szomszédosság vizsgálatakor, és akadályozza az új, élő cellák születését. A haldokló állapotnak (az élő sejtek a következő iterációban haldoklóvá válnak) köszönhetően, ebben a modellben, gyakori az irányított mozgás, tehát főleg űrhajó minták alakulnak ki. A program futtatásához ebben az esetben is elég az életjáték-modell (17. Forráskód), vagy a Seeds-modell (19. Forráskód) ciklusmagját módosítani az alábbi feltétel szerint, illetve tetszőleges kiindulási sejtmintát megadni.

20. Forráskód – Brian agya

```
17        for k = 1:N
18          for i = 2:l-1
19            for j = 2:h-1
20              B(i,j) = A(i,j);
```

```
21              count = calcNeighbor(A,i,j);
22              if A(i,j) == 2
23                B(i,j) = 0;
24              elseif A(i,j) == 1
25                B(i,j) = 2;
26              elseif A(i,j) == 0 && count == 2
27                B(i,j) = 1;
28              end
29            end
30          end
31
32          A = B;
33          plot(A);
34        end
```

3.2.3. Többdimenziós sejtautomaták

A következőkben tekintsünk meg néhány összetettebb többdimenziós modellt, amelyeket a mérnökök a mindennapokban is használnak.

3.2.e. Példa - SIR és más járvány modellek

Nevükből adódóan járványok celluláris automatákkal való modellezésre használatosak. Ezek a modellek leggyakrabban a lehetséges állapotok definiálásával adhatók meg. A járvány terjedése a szomszédos cellák tulajdonságaira vonatkozó követelményekkel módosítható. A szomszédosság vizsgálata általában Moore-módszer szerint zajlik, és minden generáció egy időegységnek felel meg. Mivel a legtöbb járvány viszonylag gyors időlefolyású, ezért sok esetben a születés és halálozás (azaz életdinamika) elhanyagolható a modell szempontjából. Egyik legegyszerűbb típusa a címben említett SIR modell, ami 0 – érzékeny *(suspectible)*, 1 – fertőzött *(infected)* és 2 – gyógyult, ellenálló, eltávolított (recovered, resistant, removed) állapotokat alkalmaz. Ez a változat figyelmen kívül hagyja az életdinamikát és szabályrendszere nem teszi lehetővé az újrafertőződést. Ebből kifolyólag járvány típusától függően több modell is létezik. Az alapmodell alkalmas egyszerű kanyaró és más hosszantartó immunválaszt kiváltó betegségek modellezésére, azonban példának okáért AIDS és COVID esetén módosításokra szorul. Bármely tetszőleges időpillanatban N

populáció értéke konstans és kifejezhető az adott állapotokban levő egyedek összegével (Kermack & McKendrick, 1921).

$$N = S(t) + I(t) + R(t) \tag{3.15}$$

Kiindulási szabályrendszer:

- Érzékeny (betegségnek kitett) cella fertőzötté válik, amennyiben legalább 2 fertőzött szomszéddal rendelkezik.
- Egy cella gyógyultnak tekinthető, ha állapota fertőzött és 4, vagy kevesebb fertőzött szomszédja van.
- Végső állapotban levő (gyógyult/halott) cella állapota nem változik.

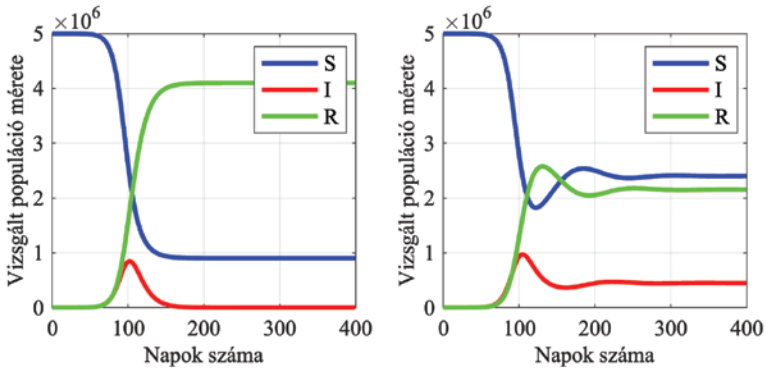
A SIR modell megvalósításához csupán három képletre van szükség. Érdemes megfigyelni, hogy a mintakódban megadott paraméterek alapján, a betegségből való gyógyulás után, az adott egyén nem fertőződhet újra ($\Delta = 0$). A delta értékének növelésével, tehát időleges immunitás szimulálásával, az eredmények látványos eltérést fognak mutatni (Frost, 2018).

$$\begin{aligned} \frac{dS(t)}{dt} &= -\beta S(t)I(t) \\ \frac{dI(t)}{d(t)} &= \beta S(t)I(t) - \gamma I(t) \\ \frac{dR(t)}{dt} &= \gamma I(t) \end{aligned} \tag{3.16}$$

21. Forráskód – SIR MATLAB kód

```
beta = 5*10^-8;   % fertõzõdés sebessége
gamma = 0.12;     % gyógyulás sebessége
delta = 0.0;      % immunitás elvesztésének sebessége
N = 5000000;      % teljes populáció N = S + I + R
I0 = 10;          % fertõzöttek száma elsõ napon
T = 400;          % vizsgált idõtartam
dt = 1/4;         % idõintervallum, lépésnagyság (6 óra)

[S,I,R] = sir_model(beta,gamma,delta,N,I0,T,dt);
tt = 0:dt:T-dt;

plot(tt,S,'b',tt,I,'r',tt,R,'g','LineWidth',2.5);
grid on;
```

```
13    xlabel('Napok száma'); ylabel('Vizsgált populáció
      mérete (5 millió lakos)');
14
15    legend('S','I','R');
16
17    function [S,I,R] = sir_model(beta,gamma,delta,N,I0,T,dt)
18
19      S = zeros(1,T/dt);
20      I = zeros(1,T/dt);
21      R = zeros(1,T/dt);
22
23      S(1) = N;
24      I(1) = I0;
25
26      for tt = 1:(T/dt)-1
27        dS = (-beta*I(tt)*S(tt) + delta*R(tt)) * dt;
28        dI = (beta*I(tt)*S(tt) - gamma*I(tt)) * dt;
29        dR = (gamma*I(tt) - delta*R(tt)) * dt;
30        S(tt+1) = S(tt) + dS;
31        I(tt+1) = I(tt) + dI;
32        R(tt+1) = R(tt) + dR;
33      end
34    end
```



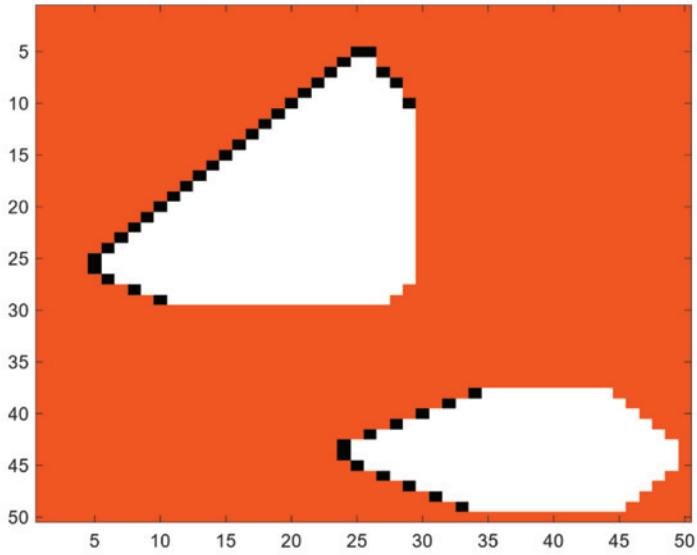
83. Ábra – SIR modell, delta = 0 és delta = 0.025 esetén

Az alap SIR modell kiegészíthető életdinamikával. Ebben az esetben új egyedek jöhetnek létre és halhatnak meg. Mivel epidemológiai szempontból sok esetben a gyógyult és halott ugyanazt jelenti (nem fertőz tovább), nem szükséges külön állapottal jelölni. Conway életjáték-modellje ehhez a modellhez is alapként szolgálhat.

22. Forráskód – SIR modell celluláris automatával

```
clear all, close all, clc

% 0 - Gyógyult/Halott (R)
% 1 - Érzékeny (S)
% 2 - Fertőzött (I)

A = ones(50,50);
l = height(A);
h = length(A);
N = 20;
B = A;

X = [25,25,26,27,27,44,44,45];
Y = [25,26,25,26,28,45,44,45];

for i = 1:length(X)
  A(X(i),Y(i)) = 2;
end

plot(A)
 for k = 1:N
  for i = 2:l-1
    for j = 2:h-1
      B(i,j) = A(i,j);
      count = calcNeighbor(A,i,j);
      if A(i,j)==1 && count >= 2
        B(i,j) = 2;
      elseif A(i,j) == 2 && count <= 4
        B(i,j) = 0;
```

```
30            end
31          end
32        end
33
34        A = B;
35        plot(A);
36        pause(0.1);
37      end
38
39
40      function piece = calcNeighbor(A,x,y)
41        piece = 0;
42        for i = -1:1
43          for j = -1:1
44            if A(x+i,j+y) == 2
45              piece = piece + 1;
46            end
47          end
48        end
49        if A(x,j) == 2
50          piece = piece - 1;
51        end
52      end
53
54      function plot(A)
55        clims = [0 2];
56        colorbar
57        colormap cool
58        imagesc(A,clims)
59        pause(0.1)
60      end
```



84. Ábra – Diszkrét térbeli SIR modell szimulációja (narancssárga – betegségnek kitett, fekete - fertőzött, fehér – gyógyult/halott)

4. DISZKRÉT ESEMÉNYŰ RENDSZEREK (DEVS)

A DEVS *(Discrete Event System Specification)*, olyan diszkrét eseményfolyamatok összessége, amelyeket bizonyos időközönként mintavétellel lehet elérni. A folytonos folyamatokkal ellentétben, itt csak egyes pontokban tudjuk lekérni a rendszer kimeneti jeleit. Ahogy a neve is sugallja, egy ilyen rendszer állapota csak akkor változik, amikor egy bizonyos esemény bekövetkezik. Azok az események, amelyek miatt a rendszer állapota megváltozik, az idő múlásával nem folyamatosan, csak bizonyos pillanatokban fordulnak elő, tehát diszkrét pillanatokban. Így egy diszkrét eseményrendszert az idő bizonyos pontjain zajló események vezérelnek. Felhasználási szempontból megemlíthetjük a Petri-hálókat, melyek működését későbbi fejezeteinkben górcső alá vesszük.

Egy rendszer általánosan leírható, mint objektumok halmaza, amelyek függenek egymástól, vagy hatnak egymásra valamilyen formában, hogy a meghatározott célokat elérjék. Eközben a rendszerre hatással lehet saját környezete. Éppen ezért kiemelten fontos a kettő – rendszer és környezet – megkülönböztetése. Ezenkívül érdemes meghatározni néhány alapvető fogalmat a diszkrét eseményű rendszerekkel kapcsolatban:

Egyed (Entity) – Egy érdeklődés tárgyát képező objektum a rendszerben, ahol ezek kiválasztása a kutatás céljától és részletességétől függ.

Tulajdonság (Attribute) – Tulajdonságok segítségével kerülnek leírásra a rendszer entitásainak (egyedeinek) jellemzői. Egyes jellemzők vonatkozhatnak kizárólag az adott egyedre, ilyen esetekben ezeket lokális értékként ajánlott kezelni.

Állapot (State) – A rendszer állapota változók halmaza, amely képes leírni az adott rendszert bármelyik adott pillanatban.

Esemény (Event) – Egy történés, ami változásokat okozhat a rendszer állapotában. Megkülönböztetünk endogén (rendszer által generált) és exogén (rendszer környezete által kiváltott) eseményeket.

Erőforrások (Resources) – Az erőforrás egy olyan egyed, ami valamiféle szolgáltatást nyújt a dinamikus entitásoknak. Az erőforrás kiszolgálhat egy vagy több egyedet egy időben, utóbbi esetben párhuzamos kiszolgálás történik. Egy dinamikus egyed igényelhet egy vagy több egységnyi erőforrást. Ha engedélyt kap az erőforrás elfoglalására, akkor bizonyos ideig marad, majd felszabadítja

a használt erőforrást. Amennyiben a kérést az erőforrás megtagadja, az egyed várósorba csatlakozik, vagy más műveletet hajt végre (pl.: átirányítják másik erőforráshoz, kilép a rendszerből). Az erőforrásoknak sok lehetséges állapota létezik. Alapesetben tétlen és foglalt állapotokkal rendelkeznek, de egyéb lehetőségek között szerepelhetnek a sikertelen, zárolt, kiéhezett állapotok is.

Lista feldolgozás (List processing) – Az egyedek kezelése, szolgáltatást nyújtó erőforrásokhoz való hozzárendeléssel történik. Ez vagy eseményjelzők csatolásával kerül megvalósításra, ami felfüggeszti a működésüket, vagy egy rendezett listára való beiktatással. Ezek a listák szimbolizálják a várólistát. A listák gyakran FIFO (first-in-first-out) módszer alapján vannak feldolgozva, de léteznek más lehetőségek is, mint például LIFO (last-in-first-out) eljárás, de lehet egy választott tulajdonság értéke alapján, vagy épp véletlenszerűen, csak hogy néhányat említsünk. Arra vonatkozóan, hogy milyen esetben lehet fontos egy adott tulajdonság értéke, jó példa lehet az SPT (shortest processing time) ütemezés. Ebben az esetben a feldolgozási idő tárolható egyedenként, mint tulajdonság. Ezután az egyedek az említett tulajdonság értéke szerint lesznek rendezve és kerülnek sorra.

Művelet és késleltetés (Activities and delays) – A művelet tulajdonképpen egy időtartam, aminek hossza ismert a művelet megkezdése előtt. Ebből következik, hogy amikor a művelet megkezdődik, a művelet vége ismert és lehetővé teszi az ütemezést. Az időtartam lehet konstans, statisztikai szóráson alapuló véletlen érték, matematikai művelet, bemeneti érték egy fájlból, vagy esemény alapján számított. A késleltetés egy határozatlan időtartam, aminek oka valamiféle rendszerállapotok kombinációja. Amikor egy egyed csatlakozik az erőforrás várólistájához, a várakozás ideje lehet kezdetben ismeretlen, hiszen az érték más bekövetkező események függvénye. Például egy fontos rendelés, vagy kritikus utasítás megelőzi a már erőforrásért sorban álló egyedeket és hamarabb kerül kiszolgálásra.

A diszkrét esemény szimulációk olyan műveleteket tartalmaznak, amelyek az idő múlását vonják maguk után. A legtöbb diszkrét esemény szimuláció tartalmaz késleltetéseket, ahogy az egyedek várakoznak. A művelet kezdete és vége, valamint a késleltetés is eseménynek számítanak (Kmeť, Modellezés és Szimuláció 2020, 2018) (Kmeť, Modellezés és szimuláció elmélete és eszközei, 2021).

1. Táblázat – Különböző rendszerek egyedei, eseményei és műveletei

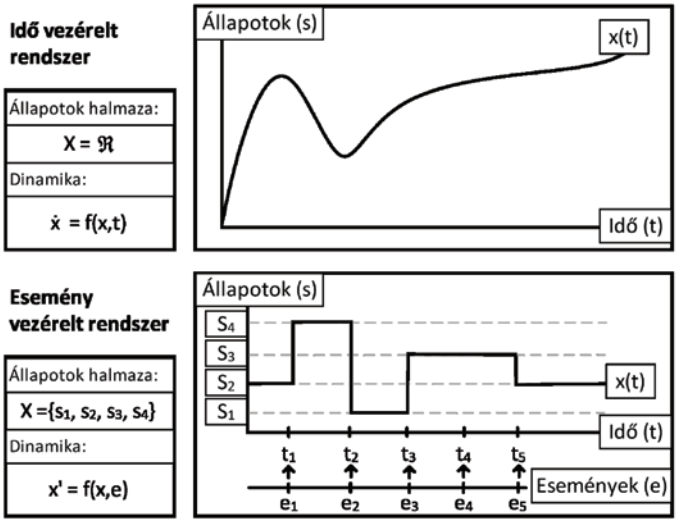
Rendszerek	Egyedek	Tulajdonságok	Művelet
Piac	Ügyfelek	Termékek	Fizetés
Telefonhálózat	Üzenetek	Hossz	Továbbítás
Bank	Ügyfelek	Egyenleg	Pénz fel-, és betétele
Forgalom	Autók	Sebesség és távolság	Vezetés
Gyártósor	Termékek	Hátralevő rendelések	Rendelések érkezése
Minőségellenőrző rendszer	Termékek	Minőség	Ellenőrzés

Determinisztikus és sztochasztikus műveletek – Amennyiben egy művelet végkimenetele teljes egészében meghatározható a bemenetek alapján, akkor determinisztikus műveletekről beszélünk. A művelet gyakran a rendszerkörnyezet része, mivel nem bármely időpillanatban ismert a pontos kimenetel. Például a gyártósoron a termékek mennyisége számos faktor függvénye: véletlen áramkimaradások, munkásstrájkok, géphiba és így tovább. Ha a művelet teljesen sztochasztikus, nincs ismert magyarázat a véletlenszerűségére. Ennek értelmében az említett kifejezések tekinthetők egymás ellentétjének. Néhány esetben, ha a művelet leírásához túl sok részlet szükséges, akkor sztochasztikusként szokták kezelni.

Endogén és exogén műveletek – Endogén műveletek azok, amelyek a rendszeren belül fordulnak elő, míg exogén műveleteknek nevezzük a környezeti műveleteket, amelyek hatással vannak a rendszerre. Ha a rendszerre nem hat exogén aktivitás, zárt rendszernek nevezzük. Amennyiben megfigyelhető exogén művelet, nyitott rendszerről beszélünk. Az előző példánál maradva, egy gyárba beérkező rendelések tekinthetők a rendszer (gyár) hatóterületén kívül esőnek, tehát része a környezetnek és így exogén műveletnek tekinthető.

Diszkért esemény szimulációs modell – Olyan modell, ami meghatározható úgy, mint ahol az állapotváltozók kizárólag időbeli diszkrét pontokon változnak, amikor az esemény bekövetkezik. Az események műveleti idők és késleltetések következményeként fordulnak elő. Az egyedek eközben versenghetnek a rendszererőforrásokért, vagy várólistákhoz csatlakoznak, amíg elérhető erőforrásra várakoznak. Művelet és késleltetés idők „feltarthatják” az egyedeket bizonyos időtartamokra. Diszkrét esemény futása szimulációs időben

(időtengelyen) történik egy olyan mechanizmus által, ami a szimulált időt előre mozgatja. A rendszer állapota frissítésre kerül minden egyes eseménynél, beleértve az erőforrások elfoglalását és felszabadítását is (Chaturvedi, 2017) (Zeigler, Muzy, & Kofman, 2019).



85. Ábra – Idő és esemény vezérelt rendszerek összehasonlítása

$$DEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$$

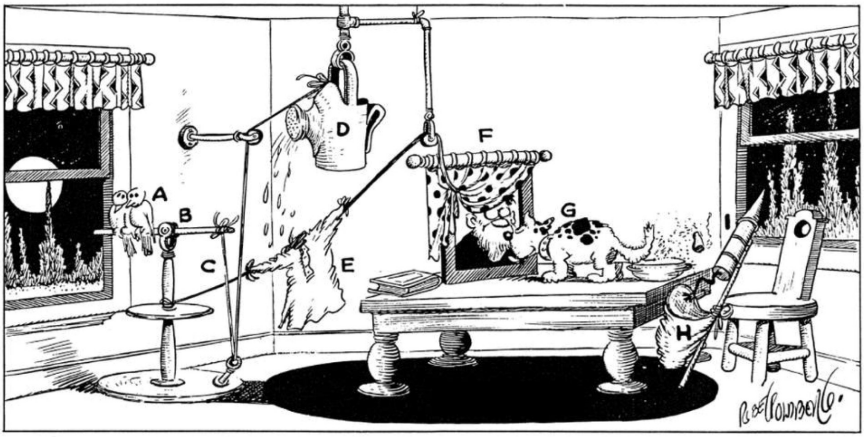
ahol:

- X – bemenetek halmaza
- Y – kimenetek halmaza
- S – szekvenciális állapotok halmaza
- δ_{ext} – $Q \times X \rightarrow S$ a külső állapotátmenet függvény
- δ_{int} – $S \rightarrow S$ a belső állapotátmenet függvény
- λ – $S \rightarrow Y$ a kimenetet leíró függvény
- ta – $S \rightarrow \mathbb{R}_0^+ \cup \infty$ az idő lépését leíró függvény
- $Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$, az összes állapot halmaza

A megfigyelt rendszer S állapotban van és a külső eseményrendszer S állapotban is marad. Az eltelt idő hossza $ta(s)$, ahol a rendszer $e = ta(s)$ időre aktiválja a $\lambda(s)$ kimenetet és ezzel megváltozik az állapot $\delta_{int}(s)$-re. Exogén (külső) esemény szintén képes állapotváltoztató hatás kifejtésére (δ_{ext}), mely esetben x külső esemény, amit s állapot és e idő határoz meg. Az e változó jelöli, hogy mennyi

időt töltött a rendszer az *s* állapotban. A belső állapotok közti átmenet kimenetet generál, míg a külső átmenetek nem generálnak kimenetet.

A jövő öntisztító hamutartója – A 86. Ábra elnagyolt hétköznapi példával igyekszik bemutatni a diszkrét ese-ményű rendszerek működését.



86. Ábra – Rube Goldberg-gép

A fényes romantikus Hold összehozza a törpepapagájokat (A) az ülőrúdon (B), ennek hatására a zsineg (C) megdönti az öntözőkannát (D) és benedvesíti a pólót (E). A póló összemegy, leleplez egy portrét (F). A kutya (G) látván gazdája arcképét csóválja a farkát és ezzel belesepri a hamut egy azbeszt zsákba (H). A parázsló csikkek meggyújtják a rakétát (I), ami kirepíti a zsák hamut az ablakon keresztül a távoli égboltra. (saját fordítás)

4.1. Tömegkiszolgáló és sorban állási rendszerek

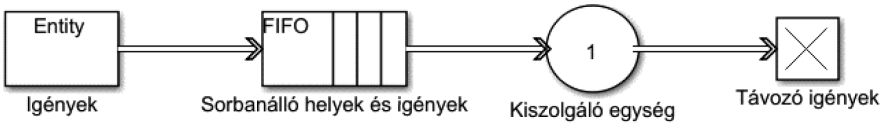
A témakör egészen Erlang (1878-1929) munkásságához vezethető vissza és a kezdeti telekommunikáció fejlődésében betöltött fontos szerepe miatt, a terminológia a mai napig az említett kutatási területen megszokott kifejezéseket használ. A tömegkiszolgálási rendszerek (röviden TKR) analitikus formában korlátozottan hoznak használható eredményeket, emiatt gyakran fordulnak közelítő eljárások és szimulációs modellek használatához.

A TKR-ek a mindennapi élet és ipar számos területén megfigyelhetők és változatosak, azonban az ilyen rendszerekben megfigyelhetők közös vonások:

- Igények beérkezése;
- A rendszer kialakítása meghatározza az igények mozgását a rendszerben;
- Kiszolgáló egységek, kiszolgálási elvek és idők fogalma;
- Sorbanállási helyek, várakozás, sorok, elutasítás;
- Kiszolgálás;
- Igények távozása a rendszerből.

TKR-ekre megfelelő példák lehetnek a telekommunikációs rendszerek, számítógépes hálózatok, tömegközlekedés (busz, vonat, repülő stb.), posta és kiszállítás, gyártórendszerek, kórházak és a szolgáltatások nagy része.

A TKR állhat több egyszerű vagy összetett tömegkiszolgáló egységből, és az ilyen rendszerek rendelkezhetnek egy, vagy több be- és kilépési ponttal.



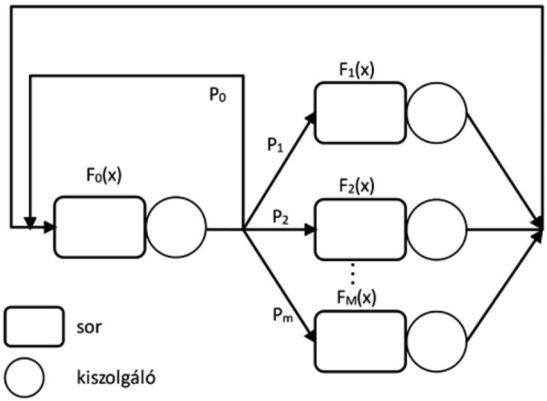
87. Ábra – Egyszerű tömegkiszolgálási rendszer SimEvents blokkjaival ábrázolva

TKR matematikai leírása:

- A beérkezési folyamat azonosítása, ami meghatározza a beérkező igényeket (igények lehetnek elvontak és rendszeren belül is létrejöhetnek).
- A rendszerbe belépő igények száma és eloszlása a rendszer állapotának függvénye.
- Az érkezési folyamatot általában, az egymás után érkező igények közti időintervallumok, valószínűségeloszlásával fejezik ki.
- Figyelembe kell venni az adott rendszer felépítését és logikáját (pl. reptéren az utazóknak át kell haladni néhány kötelező ponton).
- Kiszolgálás szabályai: FIFO, LIFO, prioritás stb.
- Figyelembe kell venni a rendszer azon mutatóit és jellemzőit, amelyek összefüggenek a kutatás céljával (pl. várakozási idő, átlagos várakozás, sorhosszúság). (Sztrik, 2009), (Györfi, Győri, & Pintér, 2002), (Szeidl, 2009)

Centrális zárt rendszer

Ez az egyik legegyszerűbb modell, mely akár számítógépek matematikai modelljének is tekinthető. A rendszerben K rögzített mennyiségű igény van jelen. A rendszer kialakításában a nyilak reprezentálják, hogy merre haladhatnak az igények. Minden kiszolgálóegység előtt elegendő sorbanálló-hely van a legfeljebb K-1 igénynek. A kiszolgálóegységek FIFO (First-in-first-out) elv alapján szolgálják ki a sorra kerülő igényeket, a kiszolgálási idők függetlenek egymástól és az i-edik egységen azonos $F_i(x)$, $0 \le i \le M$ eloszlásúak. Ha egy adott kiszolgálás befejeződik a nulladik egységen, akkor onnan az igény p_i, $0 \le i \le M (p_i \ge 0, p_0 + \ldots + p_M = 1)$ valószínűséggel átmegy az i-edik egységre, a rendszer állapotától és kiszolgálási időktől függetlenül. Amennyiben a kiszolgálás valamelyik i-edik, $1 \le i \le M$ egységen véget ér, akkor 1 valószínűséggel a 0-ik kiszolgálóegységhez kerül. A nulladik egység kiemelt szerepe miatt kapta a modell a központi (centrális) jelzőt.



88. Ábra – Egyszerű kiszolgáló rendszer

Rendszerek Kendall-féle osztályozása

A különböző gyakran használt modellek szükségessé tették, hogy kategorizáljuk őket. A szükséges modellek feladatonként eltérnek, azonban léteznek gyakran előforduló modellek, amelyek akár komplex rendszerek építőkövei is lehetnek. A rendszereket négyparaméteres szimbólum írja le röviden.

$$A \mid B \mid m \mid n$$

ahol:

A – az igények közti időközök eloszlása

B – a kiszolgálási idők eloszlása

m – kiszolgáló egységek száma (véges,végtelen)
n – várakozási helyek száma, várakozási sor hossza vagy másnéven kapacitás. Ha ez a paraméter ∞ akkor az utolsó paraméter elhagyható

Ha a sorbanálló-helyek száma $n = 0$, akkor a rendszert elutasításos rendszernek nevezzük.

A jelölés A és B változói az alábbi értékeket használhatják:
- M – exponenciális
- D – determinisztikus
- G – általános

A leggyakrabban előforduló kiszolgálási elvek

A TKR vizsgálata során, először azt kell meghatározni, hogy az adott rendszert milyen szempontból elemzik. A vizsgálat és az elemzés leggyakrabban olyan mutatók meghatározása köré épülnek, amelyek a rendszer hatékonyságával kapcsolatban információt tudnak adni. TKR-t lehetséges építeni már létező rendszer alapján és/vagy a tervezés során is hasznos lehet. Egy komplex rendszer esetén, effektivitás szempontjából, kritikus lehet a megfelelő kiszolgálási algoritmusok megválasztása.

- **FIFO** *(First in first out)* – üres kiszolgáló egység esetén az érkező igény azonnal kiszolgálásra kerül, sor esetén a kiszolgálás érkezési sorrendben történik.
- **LIFO** *(Last in first out)* – a kiszolgáló egység, a rendszerbe utoljára belépett egység kiszolgálásával foglalkozik.
- **FCFS és LCFS** *(First come first served, Last come first served)* – nagyon hasonlítanak és gyakran szinonimaként használják az előbbi két kifejezésre, azonban van eltérés a két elv között. Az esetek többségében az eredmény ugyanaz, az eltérés abban nyilvánul meg, hogy a rendszer az igények érkezési idejét figyeli.
- **Véletlen sorrend** – a sorbanállók közül a következőt, véletlenszerűen választják ki valamilyen eloszlás szerint.
- **Prioritás** – a rendszerbe érkező igényeket véges M osztályba sorolja, és az igény típusa szerint, belépésnél kap egy (prioritási) indexet. Magasabb index, magasabb prioritást jelent. A kiszolgálóegység felszabadulásakor a legmagasabb prioritású igény kerül sorra. Azonos index esetén, az előbb felsorolt elvek valamelyike segít a döntéshozatalban.

- **Abszolút prioritás** – Ha az adott igény kiszolgálási ideje alatt egy magasabb prioritású igény érkezik, a kiszolgálás azonnal megszakad, az igény visszakerül a sorba és a magasabb prioritású igény veszi át a helyét.
- **Relatív prioritás** – Előbbitől annyiban tér el, hogy nem szakad meg a folyamatban levő prioritás, viszont utána azonnal a legmagasabb prioritású következik.
- **Időosztásos kiszolgálási algoritmus** – A modern telekommunikációs rendszerek és a nagy erőforrásokkal rendelkező számítógépek alapja. Általában komplex TKR-ek esetén effektív. Az elv sajátossága, hogy lehetővé teszi a rész-kiszolgálást. Ilyenkor az igény egy töredéke kerül teljesítésre, majd visszakerül a várakozó igények közé.

G/G/1 tömegkiszolgálási rendszer leírása FIFO kiszolgálási elv mellett

A rendszer vizsgálata $t_0 = 0$ időpontban és üres kezdőpontban $L(0) = 0$ kezdődik. A sorhosszúságot kifejező $L(t)$ folyamat jobbról folytonos. Az igények a rendszerbe az egymás utáni $t_1 = x_1$, $t_2 = X_1 + X_2$ … időpontokban érkeznek és a

$$t_i = X_1 + \cdots + X_i, \quad i = 1, 2\ldots \tag{4.1}$$

időpontban beérkező i-edik igény kiszolgálási idejét Y_i jelöli. Ebből látható, hogy

$$X_i = t_i - t_{i-1}, \quad i = 1, 2\ldots \tag{4.2}$$

Ekkor, ha ismert a rendszer (X_i, Y_i), $i = 1, 2, \ldots$ ún. meghatározó sorozata, akkor ez a sorozat tetszőleges $t \geq 0$ időpontban meghatározza a rendszer állapotát, és a feladat az, hogy kiindulva az sorozatból, eljárást adjunk a rendszer különböző jellemzőinek meghatározására, tetszőleges t időpontban. Amennyiben adott a rendszer meghatározó sorozata, akkor tetszőleges $t > 0$ esetén a $[0, t)$ időintervallumban beérkező igények és onnan távozó igények száma megadható a következő mennyiséggel:

$$K = max\,\{K: t_k < t\},\ K' = max\,\{k: s_k < t\} \tag{4.3}$$

Jelölje $N(t)$, illetve $M(t)$ a t $(t \geq 0)$ időpontig a rendszerbe érkező K, illetve onnan eltávozott K' igények számát. Ekkor a t_i, $i = 1, 2,\ldots K$, illetve s_i, $i = 1, 2,\ldots K'$ felhasználásával kapjuk

$$N(t) = \sum_{k=1}^{\infty} I\,(t_k < t),\ M(t) = \sum_{k=1}^{\infty} I\,(s_k < t)$$

és innen

$$L(t) = N(t) - M(t).$$

Jelölje s_n azt az időpontot, amikor a n-edik igény kiszolgálása befejeződik, és legyen $L_n = L(s_n)$, $n \geq 1$. Jelölje Z_n azon igények számát, amelyek az n-edik igény Y_n kiszolgálási ideje alatt lépnek be a rendszerbe, azaz $Z_n = N(s_n) - N(s_n - Y_n)$, $n \geq 1$. Ekkor az $L_n = 1, 2, \ldots$ sorozatra érvényes a következő rekurrens szabály (Markov-tulajdonság) (Szeidl, 2009):

$$L_n = I(L_{n-1} > 0)(L_{n-1} - 1) + Z_n = (L_{n-1} - 1)^+ + Z_n \tag{4.4}$$

Poisson-folyamat

A független növekményű folyamatok fontos, speciális esetét képezi a Poisson-folyamat. Fontos szerepet játszik a tömegkiszolgálási rendszereken kívül számos kutatási területen, mint például az ökológia és kvantumoptika.

Az $\{N(t), t \geq 0\}$ nemnegatív értékeket felvevő, balról folytonos sztochasztikus folyamatot $\lambda(\lambda > 0)$ paraméterű, homogén Poisson-folyamatnak nevezzük, ha:

1. $N(0)=0$,
2. $N(t)$ független növekményű folyamat,
3. Tetszőleges $0 \leq s \leq t$ esetén $N(t) - N(s)$ eloszlása $\lambda(t - s)$ paraméterű Poisson, azaz

$$P(N(t) - N(s) = k) = \frac{[\lambda(t-s)]^k}{k!} \varepsilon^{-\lambda(t-s)}, \; k = 0, 1, \ldots \tag{4.5}$$

A definíció szerint az $N(t), t \geq 0, n(0) = 0$ Poisson-folyamat, olyan balról folytonos független növekményű lépcsős folyamat, melynek minden $N(t) - N(s), 0 \leq s < t$ növekménye csak nemnegatív egész értékeket vehet fel, és a növekmény eloszlása $\lambda(t - s)$ paraméterű Poisson.

Poisson-folyamatok konstrukciója

A Poisson-folyamatok konstruktív megadása, fontos szerepet játszik az elmélet és a gyakorlat (szimulációs módszerek alkalmazása) szempontjából egyaránt. Legyen $X_1, X_2, \ldots$ független 1 paraméterű exponenciális eloszlású valószínűségi változók sorozata. Vezessük be a

$$N(t) = \sum_{m=1}^{\infty} I(X_1 + \cdots + X_m < t), \; t \geq 0 \tag{4.6}$$

sztochasztikus folyamatot. Ekkor $N(t)$ homogén 1 intenzitású Poisson-folyamat. Legyen $U_1, U_2, \ldots$ független, a $(0, T)$ intervallumon egyenletes eloszlású véletlen valószínűségi változók sorozata, és legyen N ezektől független λT paraméterű Poisson-eloszlású valószínűségi változó. Jelölje

$$N(t) = \sum_{m=1}^{N} I(U_m < t),\ 0 \le t \le T \tag{4.7}$$

Ekkor $N(t)$ homogén λ intenzitású Poisson-folyamat a $[0,T]$ intervallumon.

4.2. Markov-láncok

A Markov-lánc egy matematikai rendszer, amely állapotok halmazából és az állapotok közötti átmenetek valószínűségeiből áll. Nevét Andrej Andrejevics Markov orosz matematikusról kapta, aki a 20. század elején dolgozta ki az elméletet. A Markov-láncok alapja az a feltételezés, hogy a jövőbeni állapot csak a jelenlegi állapottól függ, és független a múltbeli állapotoktól. Ezt az elvet nevezzük Markov-tulajdonságnak, vagy memória nélküli tulajdonságnak. A Markov-láncok ismertetéséhez először néhány fogalmat vezetünk be (Fewster, 2014).

Diszkrét idejű Markov-lánc: egy diszkrét idejű Markov-lánc az alábbiakból áll:

- Állapottér (S): Egy véges vagy megszámlálhatóan végtelen halmaz, amely az összes lehetséges állapotot tartalmazza.
- Átmeneti valószínűségek (P): Az S halmaz minden eleméhez $(i, j \in S)$ egy $P(i, j)$ valószínűségi értéket rendelünk, amely megadja az i állapotból a j állapotba történő átmenet valószínűségét. Ezek a valószínűségek teljesítik a következő feltételeket:
  $P(i, j) \ge 0$ minden $i, j \in S$ esetén.
  $\sum_{j \in S} P(i, j) = 1$ minden $i \in S$ esetén.

Markov-tulajdonság: Egy Markov-lánc rendelkezik a Markov-tulajdonsággal, ha a $P(X_{n+1} = i_{n+1} \mid X_n = i_n, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0) = P(X_{n+1} = i_{n+1} \mid X_n = i_n)$ feltétel teljesül minden $n \ge 0$ és $i_0, i_1, \ldots, i_n, i_{n+1} \in S$ esetén. Ez azt jelenti, hogy a X_{n+1} állapot feltételes eloszlása csak a jelenlegi X_n állapottól függ, és független az előző állapotok sorozatától.

Átmeneti mátrix: Egy $|S| \times |S|$ méretű mátrixot, amelynek elemei az egyes állapotok közötti átmenetek valószínűségeit tartalmazzák, átmeneti mátrixnak nevezünk. Jelöljük az átmeneti mátrixot P-vel. Ha P_{ij} az i állapotból a j állapotba történő átmenet valószínűségét jelöli, akkor az átmeneti mátrix elemei a következő feltételeket teljesítik:

Minden $P_{ij} \geq 0$ (minden elem nemnegatív).
Minden sor elemeinek összege 1, tehát: $\sum_{j \in S} P_{ij} = 1.$

Az átmeneti mátrix általános formája:

$$P = \begin{bmatrix} P_{11} & P_{12} & \cdots & P_{1S} \\ P_{21} & P_{22} & \cdots & P_{2S} \\ \vdots & \vdots & \ddots & \vdots \\ P_{S1} & P_{S2} & \cdots & P_{SS} \end{bmatrix}.$$

Ennek függvényében tekintsünk meg egy példát. Vegyünk egy egyszerű Markov-láncot, amelynek három állapota van: A, B, és C. Az állapotok közötti átmenetek valószínűségeit az alábbi mátrix írja le:

$$P = \begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0.4 & 0.4 & 0.2 \\ 0.3 & 0.3 & 0.4 \end{bmatrix}.$$

Ez azt jelenti, hogy például az A állapotból 0.5 valószínűséggel maradunk az A állapotban, 0.3 valószínűséggel átmegyünk a B állapotba, és 0.2 valószínűséggel a C állapotba.

Valószínűségszámítási fogalmak: Véletlen kísérletnek nevezünk minden olyan kísérletet, amely eredményeinek halmazát előre ismerjük, azonban a pontos kimenet ezek között véletlen alakul ki. A véletlen esemény – más néven kísérlet – eredménye az, amit kimenetelnek nevezünk. Egy véletlen esemény során - egy meghatározott kísérletben – elemi eseménynek nevezzük az egymást kölcsönösen kizáró lehetséges ω kimeneteleit. Az összes elemi esemény (nem üres) Ω halmazát eseménytérnek nevezzük. Ha az Ω eseménytér véges, vagy megszámlálhatóan végtelen elemből áll, akkor azt mondjuk, hogy az elemi eseménytér diszkrét. Az eseménytér lehet véges, megszámlálhatóan végtelen (diszkrét), vagy megszámlálhatatlanul végtelen (folytonos) (Hangos & Szederkényi, 1999).

Adott a nem üres megszámlálható Ω, elemi események tere (eseménytér).
$\Gamma \subset 2^{\Omega}$ – események halmaza
$P: \Gamma \rightarrow < 0;1 >$ – valószínűség eloszlás
(Ω, Γ, P) – alap valószínűségi mező

Véletlen kísérletek elvégzése során, az egyes események bekövetkeztének valószínűsége, számértékként fejezhető ki. A bekövetkező ω elemi esemény egy a véletlentől függő $X(\omega)$ számértéket eredményez.

Egy Ω-n értelmezett $X(\omega)$ valós függvényt – vagyis egy olyan függvényt, amely minden ω elemi eseményhez egy valós számot rendel – valószínűségi változónak nevezünk, ha tetszőleges x valós szám esetén $\{\omega: X(\omega) < x\} \in \Gamma$, vagyis az $\{\omega: X(\omega) < x\} \in \Gamma$ halmaz esemény. Ez utóbbi halmazt szokás az x értékhez tartozó nívóhalmaznak is nevezni.

Az X valószínűségi változó diszkrét, ha az értékkészlete véges, vagy megszámlálhatóan végtelen.

Amennyiben az X diszkrét valószínűségi változó értékkészlete $x_1, x_2, \ldots$,akkor a $p_i = P(X = x_i), i = 1, 2, \ldots$ számokat az X eloszlásának nevezzük.

Véges, vagy végtelen sok szám akkor és csakis akkor alkot diszkrét eloszlást, ha azok nem negatívak és az összegük 1.

Az $F(x) = P(x < x)$, $-\infty < x < \infty$ függvényt az X valószínűségi változó eloszlásfüggvényének nevezzük.

Diszkrét esetben $F(x) = \sum_{x_i<x} p_i, -\infty < x < \infty$.

Egy X valószínűségi változó folytonos, illetve folytonos eloszlású, ha létezik olyan nemnegatív integrálható sűrűségfüggvény, $f(x)$, $-\infty < x < \infty$, hogy tetszőleges $-\infty < a < b < \infty$ számok esetén érvényes

$$F(b) - F(a) = \int_a^b f(x)dx. \ (P(a \le X \le b)) \tag{4.8}$$

Ekkor

$$F(x) = \int_{-\infty}^{x} f(x)dx, -\infty < x < \infty \tag{4.9}$$

Azt a számot, amely körül az X valószínűségi változó megfigyelt értékeinek az átlaga ingadozik, várható értéknek nevezzük. Legyen az X diszkrét valószínűségi változó, amely véges vagy megszámlálhatóan végtelen sok $\{x_1, x_2, \ldots\}$ értéket vehet fel $p_i = P(X = x_i), i = 1, 2, \ldots$ valószínűség eloszlással. Azt mondjuk, hogy az X-nek létezik várható értéke és az $E(X) = \sum_{i=1}^{\infty} p_i x_i$

Folytonos eloszlás esetén

$$E(x) = \int_{-\infty}^{\infty} xf(x)dx \tag{4.10}$$

A Markov-lánc tranzitív, ha tetszőleges állapotból eljuthatunk tetszőleges állapotba, i.e.

$$\forall i, j \in S, p_{i,j} > 0 \tag{4.11}$$

Ha a $\{X(t)\}_{t\in T}$ tranzitív Markov-lánc, akkor létezik olyan valószínűségeloszlás π

$$\pi=(\pi_0, ...,\pi_n), \quad ahol \ \pi_i > 0, \quad i = 0, ...n, \tag{4.12}$$

amire érvényes

$$\pi = \pi P$$
$$\lim_{t\to\infty} pP^t = \pi$$

Minden p_0 kezdeti valószínűségeloszlásra.

$$B_{N\times N} = \varepsilon \begin{bmatrix} 1/N & 1/N & \cdots & 1/N \\ 1/N & 1/N & \cdots & 1/N \\ \vdots & \vdots & \vdots & \vdots \\ 1/N & 1/N & \cdots & 1/N \end{bmatrix}_{N\times N} + (1-\varepsilon)A_{N\times N} \tag{4.13}$$

Markov-tétel: Adott az $\{X(t)\}_{t\in T}$ tranzitív Markov-típusú folyamat véges S állapotokkal, akkor

$$\lim_{t\to\infty} p_j(t) = \pi_j, j = 0, ..., n, \quad ahol \sum_{j=0}^{n} p_j(0) = 1, \tag{4.14}$$

$\pi = (\pi_0, ...,\pi_n)$ eloszlás.
A lineáris egyenletrendszer megoldásai a következők:

$$\sum_{i=0}^{n} \lambda_{ij} p_i = 0, \ j = 0, ...,n \text{ és } \sum_{i=0}^{n} p_i = 1 \tag{4.15}$$

A sztochasztikus folyamatok fontos osztálya jellemezhető azzal a tulajdonsággal, hogy ha tetszőleges megadott t_0 időpillanatban ismerjük a folyamatot, akkor ez a legteljesebb információ, amit a folyamatnak megadott t_0 időpont utáni viselkedése leírásánál figyelembe kell venni a t_0 időpontig vett megfigyelésekből. Megállapítható, hogy a folyamat jövőbeli vis elkedésére a t_0 időpontig megfigyelt értékei közül, csak az az érték van hatással, amelyet a folyamat a t_0 időpontban felvesz.

Ez tekinthető egyfajta emlékezetnélküliségnek, de megfogalmazható a következőképpen is: „a folyamat jövőbeli viselkedése a jelenen keresztül függ a múlttól" (Michelberger, Szeidl, & Várlaki, 2001). Fizikai rendszerek esetén a folyamatok értékei, gyakran a rendszer állapotát adják meg a fázistérben,

amelyek megfelelnek az előbbi tulajdonságnak. Ha a folyamat valamilyen időpontban egy x értéket vesz fel, akkor elmondható, hogy a rendszer az adott időpillanatban az x állapotban tartózkodik. Az összes felvehető értéket, vagy állapotok halmazát állapottérnek nevezzük.

A Markov-láncok időparamétere és lehetséges állapotok tere lehet diszkrét és folytonos, ebből kifolyólag megkülönböztetnek diszkrét és folytonos idejű, azon belül diszkrét és folytonos állapotterű Markov-folyamatokat.

Az X folyamatot χ állapotrendszerű diszkrét idejű Markov-láncnak nevezzük, ha minden lehetséges $n = 0, 1, \ldots$ érték és tetszőleges választott $i_0, \ldots, i_n \in \chi$ állapotok mellett fennáll.

$$P(X_{n+1} = i_{n+1} \mid X_0 = i_0, \ldots, X_n = i_n) = P(X_{n+1} = i_{n+1} \mid X_n = i_n) \tag{4.16}$$

Ez az összefüggés fejezi ki azt a korábban említett heurisztikus tulajdonságot, hogy a folyamat múlttól való függése a jelenen keresztül történik.

Kijelenthető, hogy az X folyamat χ állapotterű m-ed rendű Markov-lánc ($m \geq 1$), ha minden $n = 1, 2, \ldots$ és $i_k \in \chi$, $k = 0, 1, \ldots$ mellett teljesül

$$\begin{aligned} &P(X_{n+m} = i_{n+m} \mid X_0 = i_0, \ldots, X_{n+m-1} = i_{n+m-1}) = \\ &P(X_{n+m} = i_{n+m} \mid X_n = i_n, \ldots, X_{n+m-1} = i_{n+m-1}) = \end{aligned} \tag{4.17}$$

alapján bevezethető

$$\chi' = \{(k_1, \ldots, k_m) : k_1, \ldots, k_m \in \chi\} \tag{4.18}$$

állapotterű

$$Y_n = (X_n, \ldots, K_{n+m-1}), \quad n = 0,1\ldots \tag{4.19}$$

sztochasztikus folyamatot. Amikor

$$\begin{aligned} &P(Y_{n+1} = (i_{n+1}, \ldots, i_{n+m}) \mid Y_0 = (i_0, \ldots, i_{m-1}), \ldots, Y_n = (i_n, \ldots, i_{n+m-1})) = \\ &= P(X_{n+m} = i_{n+m}, \ldots, X_{n+1} = i_{n+1} \mid X_0 = i_0, \ldots, X_{n+m-1} = i_{n+m-1}) = \\ &= P(X_{n+m} = i_{n+m}, \ldots, X_{n+1} = i_{n+1} \mid X_n = i_n, \ldots, X_{n+m-1} = i_{n+m-1}) = \\ &= P(Y_{n+1} = (i_{n+1}, \ldots, i_{n+m}) \mid Y_n = (i_n, \ldots, i_{n+m-1})) \end{aligned} \tag{4.20}$$

Ez igazolja, hogy a bevezetett $Y = \{Y_0, Y_1, \ldots\}$ folyamat elsőrendű Markov-lánc. A $p_{ij}(n, n+1) = P(X_{n+1} = j \mid X_n = i)$, $i, j \in \chi$, $n = 0, 1, \ldots$ mennyiségeket az $X = \{X_0, X_1, \ldots\}$ Markov-lánc egylépéses átmenetvalószínűségeinek nevezzük.

Az X Markov-lánc homogén, ha egylépéses átmenetvalószínűségei függetlenek az n időtől, ekkor $p_{ij}(n, n+1) = p_{ij}$, $i, j \in \chi$, $n = 0, 1, \ldots$ Ellenkező esetben a Markov-láncot inhomogénnek nevezzük.

Diszkrét egyenletes eloszlás

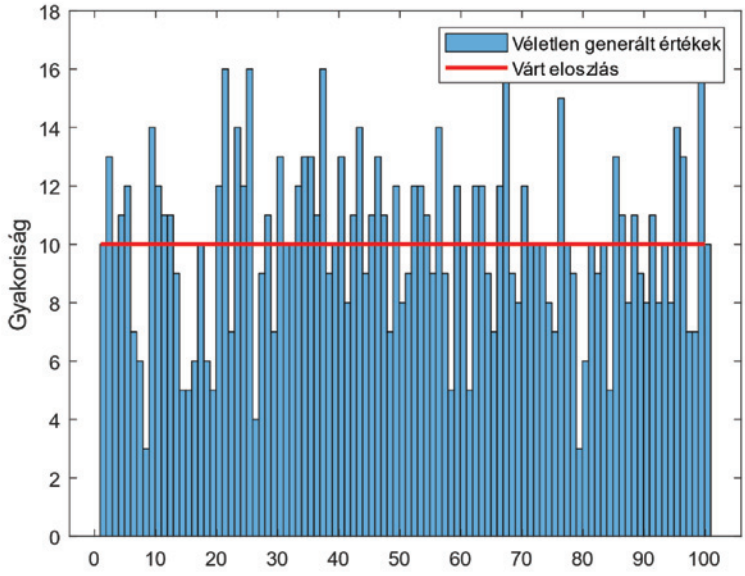
Legyen az X valószínűségi változó diszkrét, melynek a lehetséges értékei $X = \{x_1, \ldots, x_n, \ldots\}$, n valamilyen pozitív szám (lehet végtelen) és mindegyiknek azonos a valószínűsége.
X eloszlása egyenletes, ha

$$p_k = P(X = x_k) = \frac{1}{n}, 1 \leq k \leq n \tag{4.21}$$

(Fehér & Jaruska, 2015)

23. Forráskód – Egyenletes eloszlás MATLAB kódja

```
%Egyenletes eloszlás
clear all, close all
N = 1000;
uniform_min = 1;   % Minimum érték uniform eloszláshoz
uniform_max = 100; % Maximum érték uniform eloszláshoz
rng(123);
nbins = 100;

uniform_samples = unidrnd(uniform_max, 1, N);
histogram(uniform_samples, nbins);
ylabel('Gyakoriság');

hold on;
expected_counts = ones(uniform_min,uniform_max) * numel(uniform_samples) / 100;
plot(uniform_min:uniform_max, expected_counts, 'r-', 'LineWidth', 2);
legend(' Véletlen generált értékek', 'Várt eloszlás');
hold off;
```



89. Ábra – Egyenletes eloszlás

Folytonos egyenletes eloszlás

Legyen $a < b$ két tetszőleges valós szám. Azt mondjuk, hogy az X valószínűségi változó eloszlása egyenletes az (a, b) intervallumon, ha az $f(x)$ sűrűségfüggvénye létezik és fennáll

$$f(x) = \begin{cases} \dfrac{1}{b-a,} & x \in (a, b) \\ 0, x \notin (a, b) \end{cases} \qquad (4.22)$$

Binomiális eloszlás

Nézzünk meg egy n megfigyelésből álló kísérletet, melynek során azt nézzük, hogy egy bizonyos A esemény hányszor következik be. Egy kísérletben az A esemény valószínűsége p, az ellentett esemény valószínűsége $1 - p$. A kísérletet egymástól függetlenül n-szer megismételjük. Az X valószínűségi változó lehetséges értékei a kísérletsorozatban az A esemény bekövetkezésének száma. Tegyük fel, hogy

$$p = P(A),\ 0 < p < 1 \tag{4.23}$$

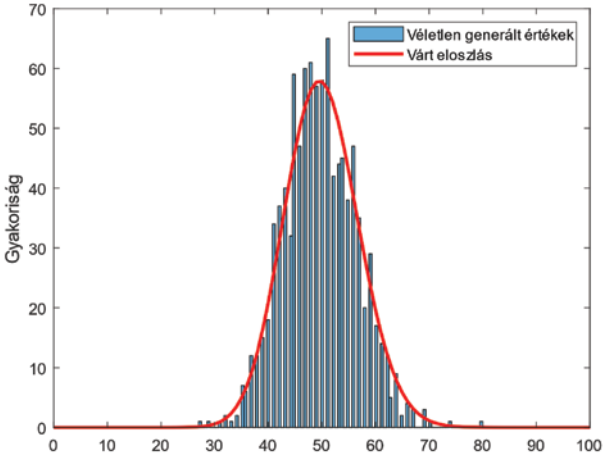
Ekkor annak valószínűsége, hogy a kísérlet során az A esemény pontosan k-szor következik be, éppen

$$p_k = P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k},\ k = 0, \ldots, n \tag{4.24}$$

(Fehér & Jaruska, 2015).

24. Forráskód – Binomiális eloszlás MATLAB kódja

```
% Binomiális eloszlás
clear all, close all
N = 1000;
binomial_dist = 0.05;    % Binomiális eloszlás
rng(123);
nbins = 100;
binomial_samples = binornd(N, binomial_dist, 1,
N);
histogram(binomial_samples,nbins);
ylabel('Gyakoriság');

hold on;
x = 0:100;
y = binopdf(x, N, binomial_dist)*N;
xlim([0,100])
plot(x, y, 'r', 'LineWidth', 2);
legend('Véletlen generált értékek', 'Várt
eloszlás');
hold off;
```



90. Ábra – Binomiális eloszlás

Geometriai eloszlás

Egy kísérletben az A esemény valószínűsége p, az ellentett esemény valószínűsége $1 - p$. Egy kísérletsorozatban figyeljük, hogy egy adott A esemény hányadik próbálkozásra következett be először. A valószínűségi változóval a próbálkozások számát jelöljük, melynek lehetséges értékei a $k = 0, 1, 2, \ldots, n, \ldots$ számok lesznek és ezek valószínűsége

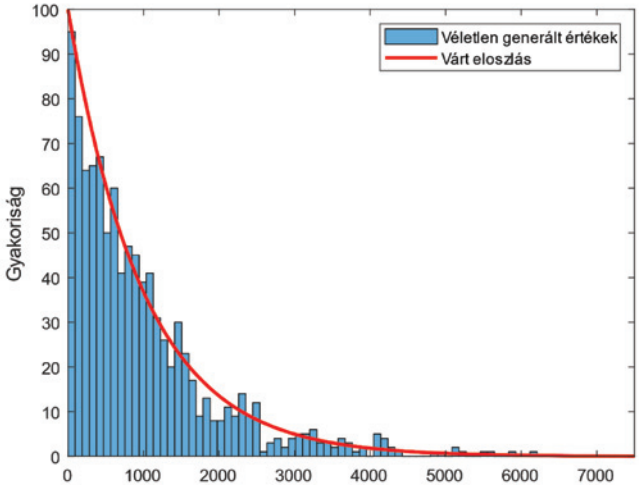
$$p_k = P(X = k) = p(1 - p)^{k-1},\ k = 1, \ldots, n \tag{4.25}$$

(Fehér & Jaruska, 2015).

25. Forráskód – Geometrikus eloszlás MATLAB kódja

```
% Geometrikus eloszlás
clear all, close all
N = 1000;
rng(123);
nbins = 100;

geometric_samples = geornd(1/N, 1, N);
histogram(geometric_samples,nbins);
```

```
9       ylabel( 'Gyakoriság' );
10
11      hold on;
12      x = 0:max(geometric_samples);
13      y = geopdf(x, 1/N)*N/10;
14      xlim([0,7500])
15      plot(x, y*N, 'r', 'LineWidth', 2);
16      legend( 'Véletlen generált értékek', 'Várt
        eloszlás' );
17      hold off;
```



91. Ábra – Geometrikus eloszlás MATLAB kódja

Poisson-eloszlás

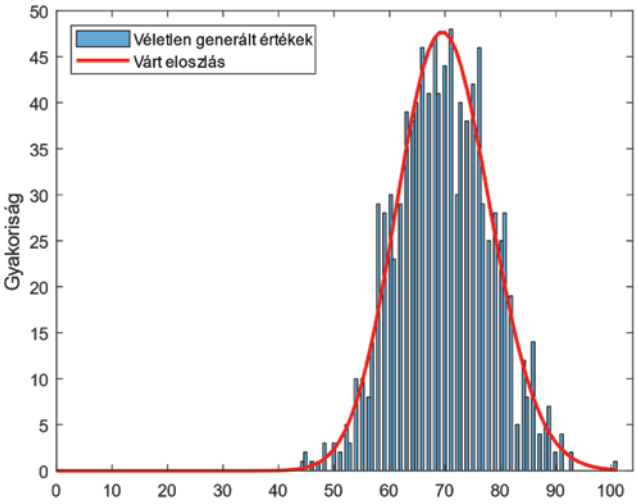
Binomiális eloszlású mennyiségek esetén, ha az n ismétlések száma növekszik, de közben a p valószínűség csökken úgy, hogy $np = \lambda$ állandó marad, akkor a binomiális eloszlás pk értékei közelíthetők a Poisson-eloszlással. A Poisson-eloszlású valószínűségi változó lehetséges értékei nemnegatív egész számok, valószínűségeloszlása pedig

$$p_k = P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}, \qquad k = 0, \ldots, n, \qquad \lambda > 0 \tag{4.26}$$

(Fehér & Jaruska, 2015).

26. Forráskód – Poisson eloszlás MATLAB kódja

```
% Poisson eloszlás
clear all, close all
N = 1000;
lambda = 70;
rng(123);
nbins = 100;

figure()
poisson_samples = poissrnd(lambda,N,1);
histogram(poisson_samples,nbins);
ylabel('Gyakoriság');

hold on;
x = 0:max(poisson_samples);
y = poisspdf(x, lambda)*N;
plot(x, y, 'r', 'LineWidth', 2);
legend('Véletlen generált értékek', 'Várt
eloszlás','Location','northwest');
hold off;
```



92. Ábra – Poisson-eloszlás

Exponenciális eloszlás

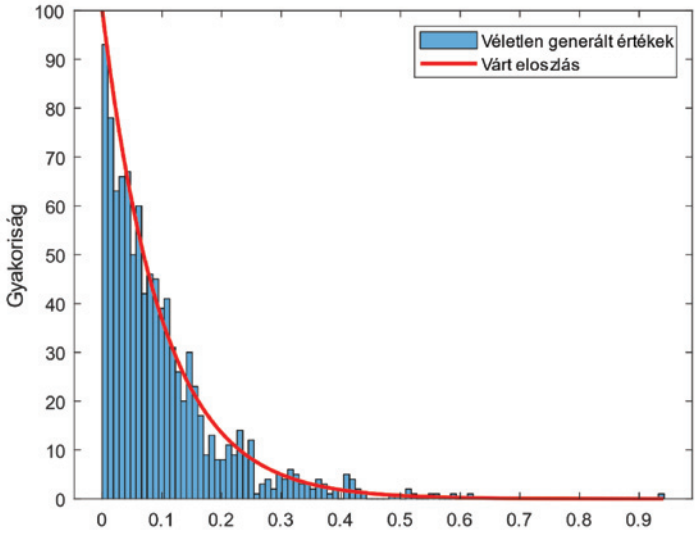
Egy valószínűségi változó exponenciális eloszlású, ha sűrűségfüggvénye és eloszlásfüggvénye

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, x \geq 0 \\ 0, x < 0 \end{cases} \tag{4.27}$$

minden x pozitív számra. Ha $x \leq 0$ az $f(x)$ és $F(x)$ függvények értékei egyenlők nullával. A λ egy tetszőleges pozitív szám, ami az eloszlás paramétere (Fehér & Jaruska, 2015).

27. Forráskód - Exponenciális eloszlás MATLAB kódja

```
% Exponenciális eloszlás
clear all, close all
N = 1000;
lambda = 10;
rng(123);
nbins = 100;

figure()
exp_samples = exprnd(1/lambda, [N, 1]);
histogram(exp_samples,nbins)
ylabel('Gyakoriság');

hold on;
x = linspace(0, max(exp_samples), N);
y = lambda*exp(-lambda*x)/5;
plot(x, y*50, 'r-', 'LineWidth', 2);
legend('Véletlen generált értékek', 'Várt
eloszlás');
hold off;

```



93. Ábra – Exponenciális eloszlás

Normális eloszlás

Egy valószínűségi változó normális eloszlású, ha sűrűségfüggvénye

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-m)^2}{2\sigma^2}} \tag{4.28}$$

m – középérték
σ – szórás

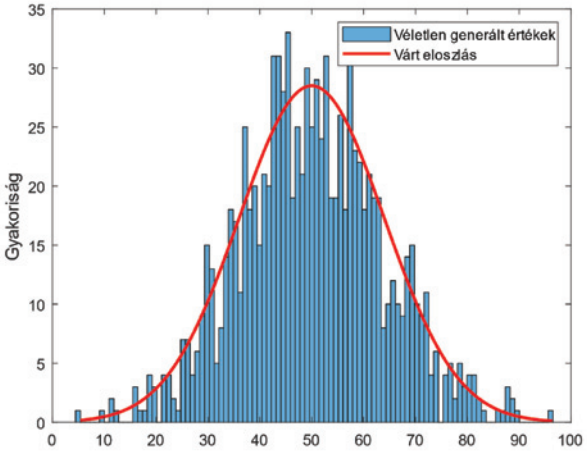
ahol m tetszőleges valós szám, és ezek a valós számok az eloszlás paraméterei. A normális eloszlású változó eloszlásfüggvénye

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-(t-m)^2}{2\sigma^2}} \tag{4.29}$$

(Fehér & Jaruska, 2015).

28. Forráskód – Normális eloszlás MATLAB kódja

```
% Normális eloszlás
clear all, close all
N = 1000;
mean = 50;
sigma = 14;
nbins = 100;
rng(123);

figure()
normal_samples = mean + sigma*randn(N, 1);
histogram(normal_samples, nbins);
ylabel('Gyakoriság');

hold on;
xlim([0,100])
x = min(normal_samples):0.1:max(normal_samples);
y = normpdf(x, mean, sigma)*N;
plot(x, y, 'r', 'LineWidth', 2);
legend('Véletlen generált értékek', 'Várt eloszlás');
hold off;
```



94. Ábra – Normális eloszlás

4.2.a. Példa – Színes labdák

Egy dobozban 2 piros, 5 sárga és 1 fehér golyó található. A dobozból 2 labdát húzunk ki.
$X(\omega)$ valószínűségi változó két értékkel:
2 – két labda azonos színű
0 – két labda nem azonos színű
Elemi események száma:
Összes eset száma $\binom{8}{2} = 28$

Két azonos színű labda húzásának valószínűsége:
Két piros labda húzásának esetek száma: $\binom{2}{2} = 1$
Két sárga labda húzásának esetek száma: $\binom{5}{2} = 10$
Két fehér labda húzásának esetek száma: $\binom{1}{2} = 0$
Az összes azonos színű esetek száma:

$$p_0 = P(X = 0) = \frac{17}{28}$$

Nem azonos színű labdák húzásának valószínűsége:
Az összes nem azonos színű esetek száma az összes esetek száma és az azonos színű esetek száma különbségeként adódik: 28 – 17 = 11

Az összes nem azonos színű esetek száma:

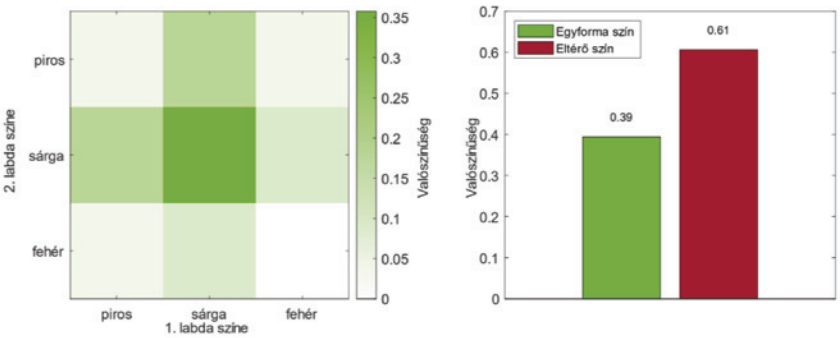
$$p_{-1} = P(X = 2) = \frac{11}{28}$$

29. Forráskód – Azonos színű labda húzása

```
% golyók színe és száma
colors = { 'piros', 'sárga', 'fehér' };
counts = [2, 5, 1];

% szimulációk száma
N = 10000;

% lehetséges kimenetelek
```

```
outcome_counts = zeros(length(colors),
length(colors));
same_color_counts = 0;
diff_color_counts = 0;

% 2 golyó véletlenszerű kiválasztása
for i = 1:N
  perm = randperm(sum(counts));

  % golyók színének meghatározása
  ball1_color = find(cumsum(counts) >= perm(1), 1);
  ball2_color = find(cumsum(counts) >= perm(2), 1);

  outcome_counts(ball1_color, ball2_color) =
outcome_counts(ball1_color, ball2_color) + 1;
  % golyók színének ellenőrzése
  if ball1_color == ball2_color
    same_color_counts = same_color_counts + 1;
else
    diff_color_counts = diff_color_counts + 1;
  end
end

% darabszámok valószínűséggé alakítása
outcome_probs = outcome_counts / num_simulations;
same_color_prob = [same_color_counts/num_
simulations, 1-same_color_counts/num_simulations];
diff_color_prob = [diff_color_counts/num_
simulations, 1-diff_color_counts/num_simulations];
% vizualizáció hőtérkép segítségével
subplot(1, 2, 1);
imagesc(outcome_probs);
cb = colorbar;
cb.Label.String = 'Valószínűség';
xlabel('1. labda színe');
ylabel('2. labda színe');
```

```
xticks(1:length(colors));
xticklabels(colors);
yticks(1:length(colors));
yticklabels(colors);

% vizualizáció oszlopdiagram segítségével
subplot(1, 2, 2);
bar(1, same_color_prob(1), 'FaceColor', '#77AC30');
hold on;
bar(2, diff_color_prob(1), 'FaceColor', '#A2142F');
xticks([]);
text(1,same_color_prob(1)+0.05,
  sprintf('%.2f',same_color_prob(1)), ...
'HorizontalAlignment', 'center');
text(2, diff_color_prob(1)+0.05,
sprintf('%.2f',diff_color_prob(1)), ...
'HorizontalAlignment', 'center');
legend({'Egyforma szín', 'Eltérő szín'},
'Location',
'northwest');
ylabel('Valószínűség');
```



95. Ábra – Hőtérkép és oszlopdiagram a példa alapján

A hőtérképen látható (95. Ábra), hogy legnagyobb esély két egyforma színű labda kihúzására sárga szín esetén van. Emellett látható az is, hogy azonos színű golyók húzásának az esélye 0.39, azaz 39%.

4.2.b. Példa - Minőségellenőrzés

A gyártási folyamat során 2 adag terméket vizsgálunk. Minden termék osztályozva van, mint megfelelő *(M)* és nem megfelelő *(N)*.

$M = 0.8$
$N = 0.2$
$X(\omega)$ – *megfelelő termékek száma*

Ω	MM	NM	MN	NN
p	$0.8 \cdot 0.8 = 0.64$	0.16	0.16	0.04
X	2	1	1	0

4.2.c. Példa - Dobókockák

Képzeljük el, hogy egy dobókockával kétszer dobunk és az eredmények összegét figyeljük. Legyen A és B az említett kockadobásokat leíró valószínűségi változók. Tételezzük fel, hogy csak az A valószínűségi változóhoz tartozó értékeket ismerjük, lehetséges-e ez alapján következtetni a B-re vonatkozó minta realizációra. Ha A az első dobás és B a második eredményét jelenti, akkor az A-t leíró információ nem biztosít B-re felhasználható információt. Tehát, ha első dobás eredménye 3, abból nem tudunk következtetéseket levonni a második dobással kapcsolatban, hiszen ezek függetlenek egymástól.

A+B	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

96. Ábra - Két független kockadobás eredményének lehetséges összegei

A / B	1	2	3	4	5	6	∑
1	1/36	1/36	1/36	1/36	1/36	1/36	1/6
2	1/36	1/36	1/36	1/36	1/36	1/36	1/6
3	1/36	1/36	1/36	1/36	1/36	1/36	1/6
4	1/36	1/36	1/36	1/36	1/36	1/36	1/6
5	1/36	1/36	1/36	1/36	1/36	1/36	1/6
6	1/36	1/36	1/36	1/36	1/36	1/36	1/6
∑	1/6	1/6	1/6	1/6	1/6	1/6	1

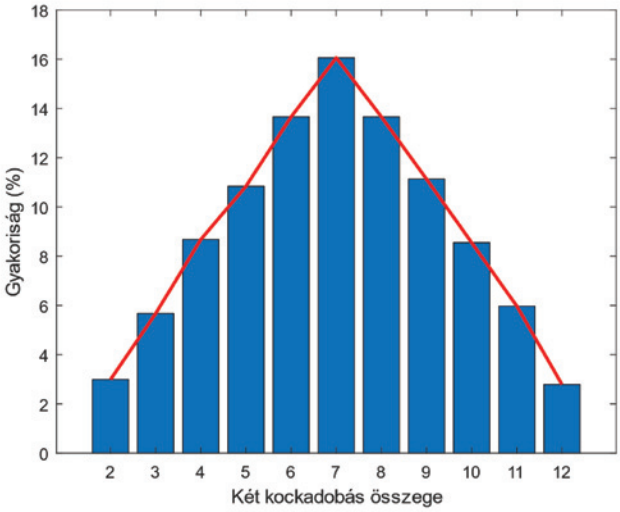
97. Ábra – Kontingencia táblázat, két független kockadobás együttes eloszlása

A 30. Forráskód tartalmazza a kódot 2 kockadobás összegeinek szimulálására. Az *N* érték jelöli a dobások számát. Az utána következő ábrán (98. Ábra) látható a kimenet 1234 véletlen kezdőérték és 10000 dobás esetén.

30. Forráskód – Kockadobások összege 2 kocka esetén

```
close all
clear all
N = 10000;    %dobások száma
K = 2;      %kockák száma
rng(1234);    %véletlen kezdőérték

val = randi([1,6],K,N);
sum_val = sum(val);

%kockadobások összegének gyakorisága
counts = zeros(1,11);
for i = 2:12
  counts(i-1) = sum(sum_val == i);
end

%y tengely értékei
percentages = counts / N * 100;

x = 2:12;
bar(x, percentages);
```

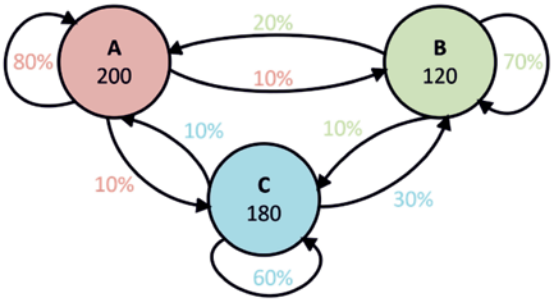
```
21        xlabel('Két kockadobás összege');
22        ylabel('Gyakoriság (%)');
23
24        hold on
25        mean_val = mean(sum_val);
26        line(x, percentages, 'Color', 'r', 'LineWidth',
          2);
```



98. Ábra – Eloszlás 2 kockával való dobás esetén

4.2.d. Példa - Diszkrét Markov-lánc feladat megoldása

Adott 500 személy, akiknek az étkezési szokásait vizsgálja és írja le a modell. A kezelhetőség és átláthatóság kedvéért ugyanabból a 3 opcióból választhatnak minden nap. A 3 opció 3 menüt takar, amelyeket az „A”-val, „B”-vel és „C”-vel jelölünk. Az egyéni preferenciák nem ismertek, azonban tudjuk, hogy aki az adott menüt választotta, milyen esélyekkel fog választani a következő nap. A 3 opcióból következően bármelyik pontból 3 opció áll rendelkezésre, választhatják ugyanazt a menüt, vagy valamelyiket a fennmaradó kettőből. Az „A” menüt választók esetén az esélyek a következőképp alakulnak: 80%-uk ismét A menüt fogja választani, míg a fennmaradó emberek fele-fele arányban (10-10%) kérik a másik menüket.



99. Ábra – A feladathoz tartozó Markov-lánc vizualizálása

Átmenetvalószínűség mátrix

Legyenek a_1, a_2, a_3,..., a_n egy Markov-lánc állapotai és P egy olyan $n \times n$-es mátrix, melynek minden p_{ij} eleme az a_j állapotból állapotba való áttérés átmenetvalószínűségét jelöli:

$$P = \begin{bmatrix} p_{11} & \cdots & p_1 \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nn} \end{bmatrix} \tag{4.30}$$

A példa esetében:

$$P = \begin{bmatrix} 0.8 & 0.2 & 0.1 \\ 0.1 & 0.7 & 0.3 \\ 0.1 & 0.1 & 0.6 \end{bmatrix}$$

Érdemes megfigyelni, hogy a mátrix bármelyik függőleges oszlopának összege 1. Az 500 alany eloszlása a vizsgált folyamatban:

$$\mathrm{A} = \frac{200}{500} = 0.4 = 40\%$$
$$\mathrm{B} = \frac{120}{500} = 0.24 = 24\%$$
$$\mathrm{C} = \frac{180}{500} = 0.36 = 36\%$$
$$40\% + 24\% + 36\% = 100\%$$

Az átmenetvalószínűség mátrix kiegészítése a létszám szerinti súlyozással a következőt adja:

$$\begin{bmatrix} A_1 \\ B_1 \\ C_1 \end{bmatrix} = \begin{bmatrix} 0.8 & 0.2 & 0.1 \\ 0.1 & 0.7 & 0.3 \\ 0.1 & 0.1 & 0.6 \end{bmatrix} \cdot \begin{bmatrix} 0.4 \\ 0.24 \\ 0.36 \end{bmatrix} =$$

A mátrixszorzás szétírása és elvégzése után az alábbi eredményt hozza:

$$\begin{bmatrix} 0.8 \cdot 0.4 + 0.2 \cdot 0.24 + 0.1 \cdot 0.36 \\ 0.1 \cdot 0.4 + 0.7 \cdot 0.24 + 0.3 \cdot 0.36 \\ 0.1 \cdot 0.4 + 0.1 \cdot 0.24 + 0.6 \cdot 0.36 \end{bmatrix} = \begin{bmatrix} 0.404 \\ 0.316 \\ 0.280 \end{bmatrix}$$

A kapott vektor összetevőinek összege 1, ami gyors visszajelzést adhat a számítás helyességéről és a kijött értékek értelmében az A_1, B_1, C_1 értékek így alakulnak:

$$A_1 = 0.404 \cdot 500 = 202$$
$$B_1 = 0.316 \cdot 500 = 158$$
$$C_1 = 0.280 \cdot 500 = 140$$

Feladat megoldása MATLAB segítségével

31. Forráskód – Markov-lánc diszkrét esete

```
        clear all, close all, clc
1       % Markov-lánc Diszkrét eset
2       % 1 - átmenetek száma
3       % x - átmenet mátrix
4       % y - kezdeti értékek
5       % y1 - eredmény
6
7       x = [0.8 0.2 0.1
8          0.1 0.7 0.3
9          0.1 0.1 0.6]
10
11      y = [0.40;
12         0.24;
13         0.36]
14
15
16      hold on
```

```
17      plot(0,y, '*r ')
18      y1 = x * y
19      plot(1,y1, '*r ')
20      hold off
```

A megjelenített eredményeknél, érdemes csoportosítani az összetartozó értékeket, így növelve az olvashatóságot. Ez kis minta esetén a grafikon beállításaiban is könnyen kivitelezhető. Az ábrán *x = 0* ponton láthatóak kiindulási értékek és *x = 1* gyűjti az 1. átmenet után kapott új eredményeket. Piros szín *A*, kék *B* és C pedig sárgával van szimbolizálva.

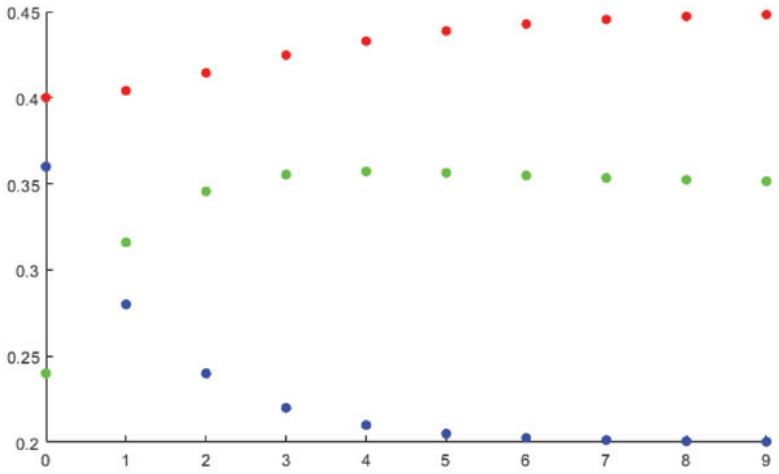
Markov-lánc feladatmegoldása több átmenet esetén

Több *N* átmenet kiszámolása az előbb bemutatott kézi módszerrel időigényes, azonban az előző kód kibővíthető *N* számú átmenet kezelésére.

32. Forráskód – Markov-lánc N=9 átmenet esetén

```
1       clear all, close all, clc
2       % Markov-lánc Diszkrét eset
3       % N - átmenetek száma
4       % x - átmenet mátrix
5       % y - kezdeti értékek és eredmény
6
7       x = [0.8 0.2 0.1
8          0.1 0.7 0.3
9          0.1 0.1 0.6]
10
11      y = [0.40;
12         0.24;
13         0.36]
14
15      N = 9;
16      hold on
17
18      for i = 0:1:N
19        plot(i,y(1), '.r ', 'MarkerSize' ,20)
20        plot(i,y(2), '.g ', 'MarkerSize' ,20)
```

```
21          plot(i,y(3), '.b ', 'MarkerSize ',20)
22          y = x*y
23        end
24        hold off
```



100. Ábra – Markov-lánc vizualizálása 9 átmenet esetén, ahol az X tengely értékei egy átmenetet jelölnek

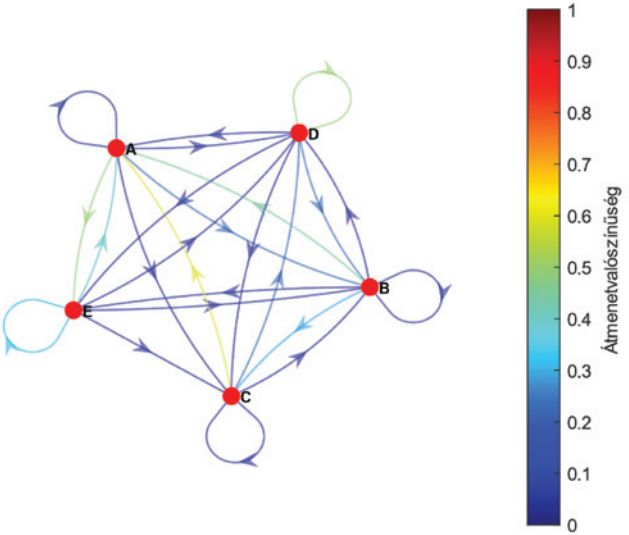
A Markov-láncok fejlett vizualizációját teszi lehetővé az Econometrics Toolbox. A valószínűségmátrix kényelmesen megadható mátrix formában, és a graphplot () függvény meghívásával elkészül a gráf.

Az alábbi *P* átmenetvalószínűség mátrix által értelmezett Markov-lánc vizualizálása a következőképpen történhet:

$$P = \begin{bmatrix} 0.1 & 0.2 & 0.1 & 0.1 & 0.5 \\ 0.1 & 0.3 & 0.1 & 0.6 & 0.1 \\ 0.1 & 0.2 & 0.1 & 0.2 & 0.1 \\ 0.5 & 0.5 & 0.1 & 0.0 & 0.1 \\ 0.4 & 0.1 & 0.1 & 0.1 & 0.3 \end{bmatrix}$$

33. Forráskód – Econometrics Toolbox segítségével létrehozott vizualizáció

```
statenames = [ "A " "B " "C " "D " "E "];
P = [0.1 0.2 0.1 0.1 0.5
   0.5 0.1 0.3 0.1 0.1
   0.6 0.1 0.1 0.2 0.0
   0.1 0.2 0.1 0.5 0.1
   0.4 0.1 0.1 0.1 0.3];
mc = dtmc(P, 'StateNames', statenames);
mc.P
sum(mc.P,2)

figure;
graphplot(mc, 'ColorEdges',true);
cb = colorbar;
cb.Label.String = 'Átmenetvalószínűség';
```



101. Ábra – Markov-lánc vizualizálása MATLAB segítségével

4.2.1. Page Rank

A Markov-lánc kiválóan alkalmas page ranking elemzésére. Természetéből adódóan maga a lánc egy gráf, ahol egyes állapotokból mozgás megy végbe a többi állapot felé. Ennek az algoritmusnak a fejlettebb változatát használják, az egyes internetes oldalak fontosságának felmérésére.

Jelöljük $\pi = \lim_{m\to\infty} pA^m$, ahol $p=\left(\frac{1}{N}, ..., \frac{1}{N}\right)$.

Ha $\pi = \pi A$, akkor a $\pi=(\pi_1, ..., \pi_N)$ vektort fontossági vektornak nevezzük (rank-vector). Az *'i'* weboldal értéke π_i.

$$BN \times N = \varepsilon \begin{bmatrix} 1/N & 1/N & 1/N & 1/N \\ 1/N & 1/N & 1/N & 1/N \\ \vdots & \vdots & \vdots & 1/N \\ 1/N & 1/N & \cdots & 1/N \end{bmatrix} + (1-\varepsilon)A_{N\times N}$$

N – weboldal
$A_{N\,x\,N}$ – mátrix
Az *'i'* weboldalon n link van
$A(i, j) = \{\frac{1}{n}, \quad j \leftarrow i\}$
Akkor érvényes $\sum_{j=1}^{N} A(i, j) = 1, A(i, j) \geq 0$.
ahol:

- r – PageRank értékek vektora
- ε – tompító skaláris faktor,ami azt simulálja, hogy $1 - \varepsilon$ valószínűséggel elugrunk bármelyik oldalra egyenletes eloszlás szerint választva
- A' – szomszédsági mátrix transzponáltja
- d – vektor,ami tartalmazza az egyes egységek kimeneti fokát
- n – egységek száma
- s – PageRank értékek összege linkkel nemrendelkező weboldalaknak

4.2.e. Példa - Page rank

Az alábbi példa 4 weblap (a, b, c, d weblapok) közti kapcsolatokat vizsgálja és ábrázolja. Az „a" weblap esetében minden állapotátmenet egyenlő esélyekkel rendelkezik, azonban a többi weblap, különböző átmenetvalószínűségekkel dolgozik, amelyek a következők:

a → a, 25%	b → a, 50%
a → b, 25%	c → b, 100%
a → c, 25%	d → c, 33.3%

a → d, 25% d → a, 33.3%
b → d, 50% d → b, 33.3%

2. Táblázat – Weblapok közti átmenetvalószínűségek leírása

	a-ból	b-ből	c-ből	d-ből
a-ba	25%	50%		33.3%
b-be	25%		100%	33.3%
c-be	25%			33.3%
d-be	25%	50%		

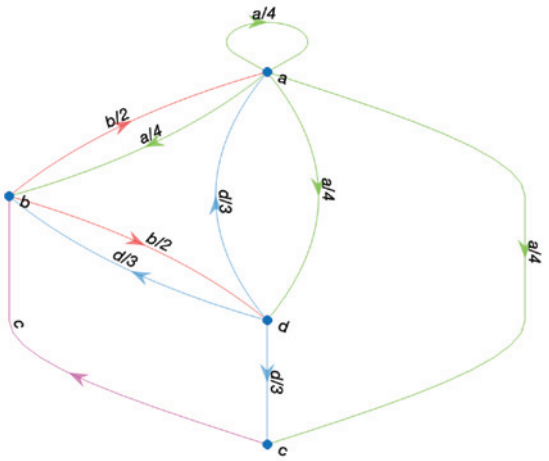
	a-ból	b-ből	c-ből	d-ből
a-ba	a/4	b/2		d/3
b-be	a/4		c	d/3
c-be	a/4			d/3
d-be	a/4	b/2		

34. Forráskód – PageRank értékek számítása 4 weblap esetén

```
s = { 'a' 'a' 'a' 'a' 'b' 'b' 'c' 'd' 'd' 'd '};
t = { 'a' 'b' 'c' 'd' 'd' 'a' 'b' 'c' 'a' 'b '};

G = digraph(s,t);

labels = { 'a/4 ' 'a/4 ' 'a/4 ' 'a/4 ' 'b/2 ' 'b/2 ' 'c '
 'd/3 ' 'd/3 '
  'd/3 '};
p = plot(G, 'Layout ' , 'layered ' , 'EdgeLabel ' ,labels);
highlight(p,[1 1 1 1],[1 2 3 4], 'EdgeColor ', 'g' )
highlight(p,[2 2],[1 4], 'EdgeColor ', 'r' )
highlight(p,[3],[2], 'EdgeColor' , 'm' )

pr = centrality(G, ' pagerank ' , ' FollowProbability '
,0.85)
```

A centrality() függvény segítségével kiszámolható az egyes weblapok PageRank értéke. Megadott példaértékek esetén a következő eredményt adják:

$$a = 0.2963,\ b = 0.3069,\ c = 0.1659,\ \mathrm{d} = 0.2309$$



102. Ábra – PageRank algoritmus vizualizálása

4.3. Sorbanállási rendszerek

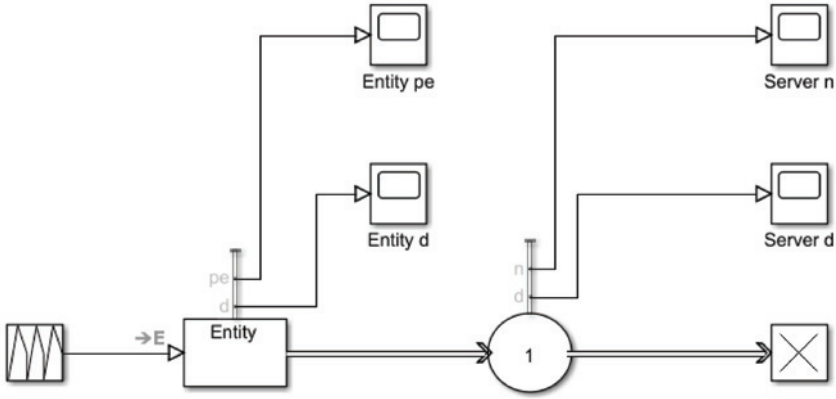
A sorbanállási rendszerek a mindennapi életben előforduló várakozások vizsgálatával foglalkoznak. Ahhoz, hogy a teljesség igényével jellemezzünk egy sorbanállási rendszert, „azonosítanunk kell azt a sztochasztikus folyamatot, amely a beérkező igényeket írja le, és meg kell adnunk a kiszolgálás szabályait és struktúráját. A beérkező folyamatot általában az egymás után beérkező igények közötti időintervallumok, mint valószínűségi változók eloszlásának segítségével jellemezhetjük." (Sztrik, 2009, old.: 12.) Szükséges továbbá megadni a kiszolgálási időt (sztochasztikus mennyiség), amely a beérkező igények kiszolgáló egységekkel szemben támasztott követelményeinek nagysága. A kiszolgálás ideje annak az időintervallumnak a hosszát jelenti, amelyet az igény a kiszolgálóegységben eltölt. A kiszolgálás szabályára és struktúrájára vonatkozóan, további mennyiségeket (jellemzőket) kell meghatározni. Ilyen jellemző a rendelkezésre álló kiszolgálóegységek száma, valamint a befogadóképesség, ami nem más, mint a kiszolgálóegységben és a várakozási sorban tartózkodó igények maximális száma, amit gyakran végtelennek tekintünk. A kiszolgálási sorrend írja le azt a szabályt, amely szerint a várakozók közül sorra kerülnek

az egyes igények, kiszolgálás céljából. A leggyakrabban használt kiszolgálási elvek: FIFO (First In, First Out) érkezési sorrendben; LIFO (Last In, First Out) fordított sorrendben történő kiszolgálások. Ha a beérkező igényeket bizonyos csoportokba tartozás szerint meg lehet különböztetni, akkor a csoportok között prioritást lehet megállapítani, és ezen a prioritáson alapul a kiszolgálás sorrendje. A sorbanállási rendszerek hatékonyságának és teljesítményének vizsgálatához a következő rendszerjellemzőket igyekszünk meghatározni: az igények várakozási ideje; a rendszerben levő igények száma; a foglaltsági intervallum hossza (vagyis az a folytonos időintervallum, amelyben a kiszolgáló egység állandóan foglalt); az üresjárati időszakasz hossza; a pillanatnyi munkahátralék eloszlása (Sztrik, 2009).

4.3.a. Példa - Entitások periodikus létrehozása

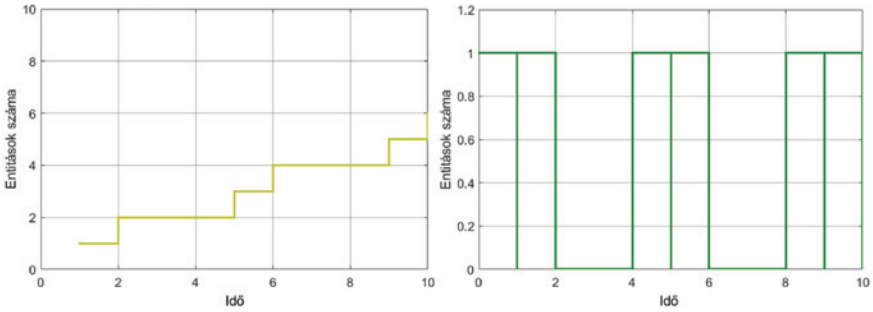
A 103. Ábra, egy periodikusan ismétlődő jelet használó modellt mutat be az entitások generálására. Ennek kivitelezéséhez az entitások létrehozásáért felelős blokk idő forrását ***(Time source)*** jel bementre ***(Signal port)*** szükséges állítani, és a Simulink könyvtárból az előző alfejezetben bemutatott ***Repeating Sequence*** blokkra lesz szükség. Utóbbi paraméterei a következők:
Time values:[0 1 2]
Output values:[1 2 3]

pe – blokkban jelen lévő függő entitások
d – blokkot elhagyó entitások száma
n – blokkban lévő entitások száma
w – átlagos várakozási idő



103. Ábra – Periodikusan generált entitások kiszolgálása

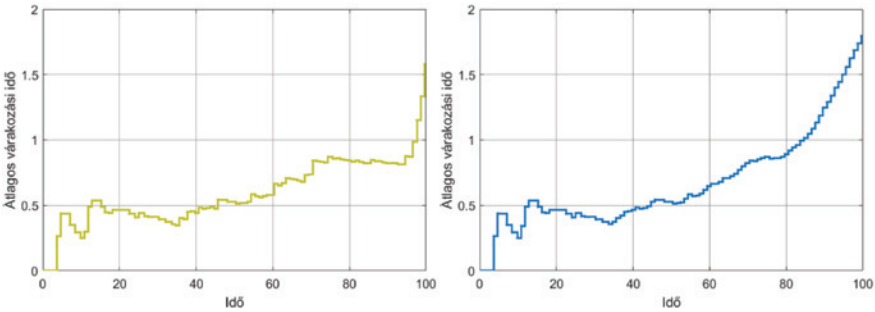
Az entitásgeneráló és entitáskiszolgáló blokkok grafikonja, ami a d blokkból való kilépést *(departure)* hivatott megjeleníteni, ebben az esetben egy időegységnyi eltolódással ugyanazt az eredményt fogja megjeleníteni. A megadott periodikus minta okozta egyedgenerálás-felfüggesztés és az entitáskiszolgáló használtságában jelentkező üresjárat megfigyelhető a következő két grafikonon.



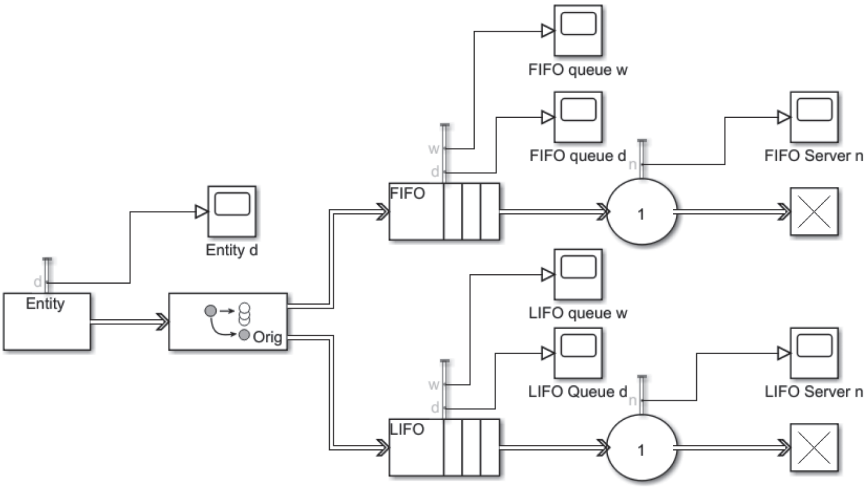
104. Ábra – Entitáskiszolgáló (Entity Server) elhagyása (sárga) és használtsága

4.3.b. Példa - FIFO és LIFO sorbanállási módok összehasonlítása

A következő modell képes szemléltetni a különbségeket FIFO és LIFO sorakozás között. A létrehozott entitásokból másolat készül, ezzel elérve, hogy az entitások párhuzamosan haladjanak a két kialakított ágon. A paraméterek további finomítása nélkül is látszik, hogy az entitásokkal haladva, az átlagos kiszolgálási idők a két ágon kezdenek jelentősen eltérni. FIFO modell esetén a várakozási idők viszonylag egyformák voltak, LIFO modellt alkalmazva viszont meredekebb ugrások figyelhetők meg.



105. Ábra - FIFO (kék) és LIFO (sárga) sorakozás átlagos ideje



106. Ábra – Alapmodell FIFO és LIFO sorbanállás összehasonlítására

4.3.c. Példa - Erőforráskezelés megfigyelése

A SimEvents rendszer kiválóan alkalmas változatos erőforrások kezelésének megfigyelésére. Ilyen szimulációk során, hasznos információkat lehet kideríteni az adott erőforrás használatával, fogyásával kapcsolatban.

Tételezzük fel, hogy van két diák (entitás), akik dolgoznak. A számukra kitűzött feladat igényel egyrészt manuális munkát – amikor füzetbe írnak –, másrészt számítógéppel végrehajtott bonyolult számításokat. A két megfigyelt diák eltérő képességekkel rendelkezik. Egyikük több időt szán a kézzel végzett munkára és viszonylag jól meghatározható a számítógéphasználata. Ezzel szemben a második tanuló kevesebb időt fordít jegyzetelésre és a számítógépes munka időtartama szélesebb intervallumon változik. A tanulók munkáját az alábbi képlet és egyenletes eloszlás jellemzi:

$$dt = m + (M - m) \cdot rand \tag{4.31}$$

Első tanuló paraméterei

- Manuális számítás: $m = 20, M = 30$
- Számítógéphasználat: $m = 10, M = 15$

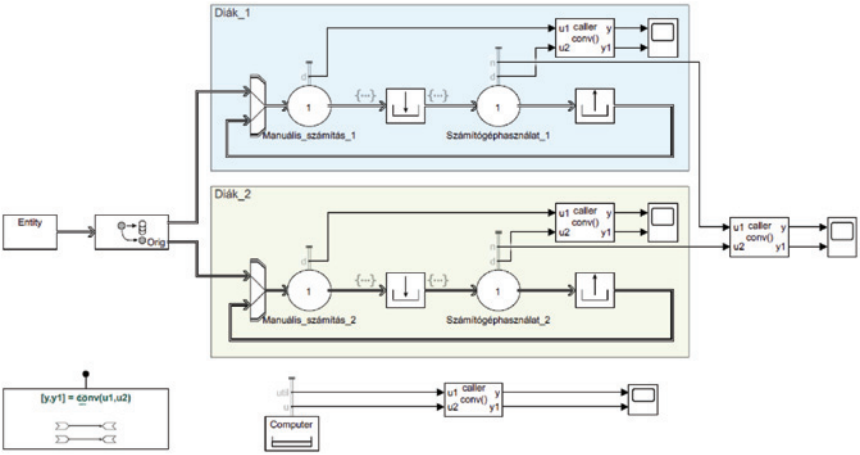
Második tanuló paraméterei

- Manuális számítás: $m = 15, M = 20$
- Számítógéphasználat: $m = 5, M = 25$

A rendelkezésre bocsátott adatok alapján, meghatározhatók a szükséges kiszolgálóblokkok paraméterei. Minden egyes blokk kapacitása 1, a kiszolgálási idő pedig egy MATLAB esemény, amely a megadott minimum (*m*) és maximum (*M*) értékek beillesztésével definiálható a forráskódban.

35. Forráskód – Egyenletes eloszlású véletlen szám

```
1       persistent rngInit;
2       if isempty(rngInit)
3         seed = 12345;
4         rng(seed);
5         rngInit = true;
6       end
7
8       % Minta: Egyenletes eloszlás
9       % m: Minimum, M: Maximum
10      m = minimum; M = maximum;
11      dt = m + (M - m) * rand;
```



107. Ábra – Erőforrás használatának megfigyelésére épített zárt rendszer

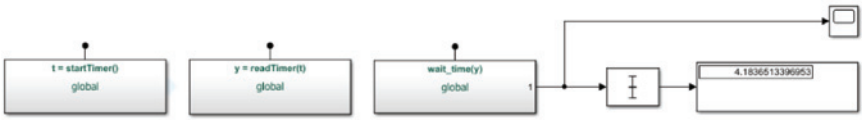
A rendszer megépítése után, létre kell hozni az idő méréséért felelős függvényeket, és felruházni az entitásokat az ennek használatához nélkülözhetetlen tulajdonsággal. A példában a számítógép a limitált erőforrás, szimuláció során 1 darab áll belőle rendelkezésre, és mivel munka befejeztével újra használható,

megújulónak tekinthető. Az erőforrást a Resource Pool Student nevű blokk szimbolizálja.

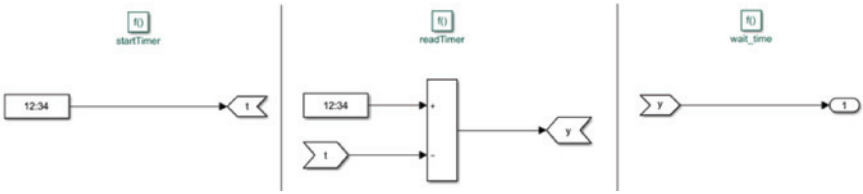
A példa során pontosan 2 entitás aktivitását vizsgáljuk. Ezek létrehozására több megoldás is létezik: 2 entitást hoz létre a generátor, és váltó **(Entity Input Switch)** segítségével a számukra kijelölt rendszerbe kerülnek, 2 generátor egymástól függetlenül hoz létre 1-1 entitást, vagy – ahogyan azt a 107. Ábra mutatja – egy entitást hoz létre a rendszer, amit az **Entity Replicator** blokk lemásol. A feladatvégzés idejét a szimuláció futási ideje adja meg, így nincs szükség az entitások megszüntetésére. Amíg tart a szimuláció, addig az egyedek a számukra izolált részen periodikusan végzik a manuális munkát, esetlegesen várnak a számítógéphasználatra és/vagy használják a számítógépet.

A két diák közötti átfedést a közösen használt erőforrás jelenti, amihez mindketten hozzáférnek. Az erőforrás használatát két tevékenység jelzi, először az entitás megszerzi a számítógépet a **Resource Acquirer** blokkon keresztül, majd a munka végével „vissza kell adnia" az erőforrást. Ezt az előző blokk párja, név szerint a **Resource Releaser** végzi. Mindkét blokk használata esetén meg kell adni a létrehozott és használni kívánt erőforrást a kiválasztott erőforrások (Selected Resources) részben.

Ezek beállítása után következik az idő mérését végző függvények létrehozása. Három globális függvényre lesz szükség, amelyek mind létrehozhatók a **Simulink Function** segítségével. Saját függvények létrehozásánál célszerű először magát a függvényt megadni (blokkon belül felső zöld színű sor), mivel ez fogja meghatározni a függvényen belőle létrejövő objektumokat, illetve ezen a néven lehet majd elérni.



108. Ábra – Erőforrás használati idejét mérő függvények



109. Ábra – Függvények belső szerkezete

Az első, **startTimer()** függvény kizárólag annyi feladatot lát el, hogy adott pillanatban elmenti az időt a t nevű változóba. A második, **readTimer()** nevű függvény, az aktuális időből kivonja az első függvény által megszerzett *t* értéket, így megkapva a számítógép használatának pontos időtartamát. Az utolsó függvény (**wait_time()**) lehetővé teszi az említett érték kiolvasását és vizualizálását. Végezetül a **readTimer()** meghívásáról kell gondoskodni, ez lehetséges a forrás megszerzéséért felelős blokk (Resource Acquirer) paraméterei között. Az egyed blokkba történő belépésekor (Event actions - Entry) lezajlik az óra indítása, míg kilépésnél az eltelt idő kiszámítása.

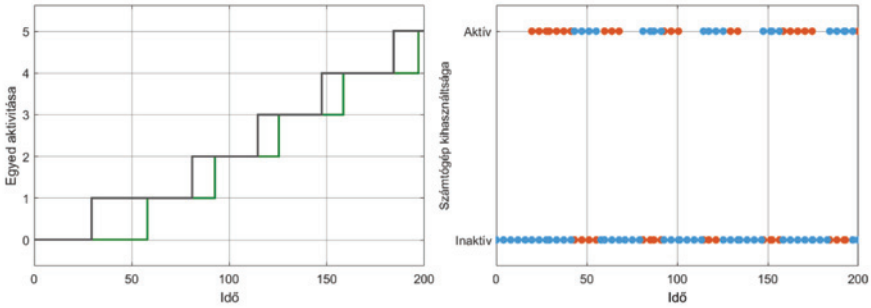
36. Forráskód – Resource Acquirer blokkba belépés (Entry) és blokk elhagyás (Exit) eseménye

```
% Block Parameters: Resource Acquirer
% Event actions - Entry
entity.Timer = startTimer();
```

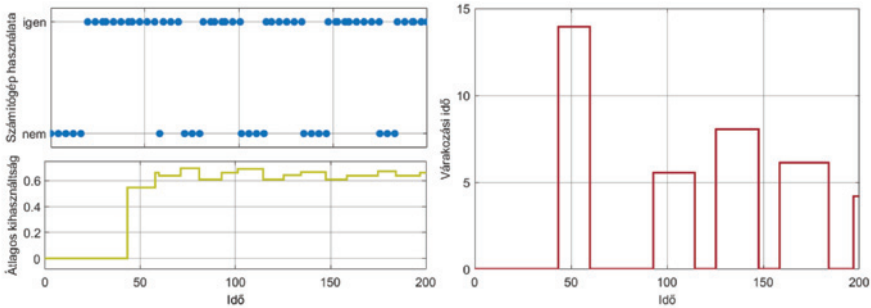
```
% Block Parameters: Resource Acquirer
% Event actions - Exit
elapsedTime = readTimer(entity.Timer);
wait_time(elapsedTime);
```

Mivel a Scope alapértelmezetten nem képes egyszerre több érték megjelenítésére, a bemenetek számának megadása ellenére sem, a megjeleníteni kívánt adatokat szükséges először egy függvényen átfuttatni. Az említett függvényt a 107. Ábra alkalmazza és utána a Function Caller nevű blokk beiktatásával és rajta a kért adatok átvezetésével már megjeleníthetők az értékek. Fontos, hogy a Function Caller blokk Function Prototype paramétere a függvény nevét viselje, ebben az esetben *[y, y1] = conv(u1, u2)*.

A rendszer futási ideje 200 időegység volt, ami a megadott paraméterek mellett $y = 4.18365$ egységnyi átlagos várakozási időt jelentett a rendszer 2 entitása számára.



110. Ábra – Első egyed aktivitása, ahol szürke a manuális munkát, zöld a számítógéppel végzett munkát jelöli (balra). A számítógép használata 200 időegység alatt entitások szerint bontva (jobbra).



111. Ábra – Számítógép használata (kék) és átlagos használata (sárga) (balra) és erőforrásra való várakozással töltött idő (jobbra)

4.3.d. Példa - Erőforráskezelés megfigyelése 3 entitás esetén

Az előző alfejezetben bemutatott modell kiegészítésével, egy sokkal kifinomultabb rendszer válik elérhetővé. Ebben a példában, másolás segítségével, egy újabb entitást hozunk létre, aki szintén versengeni fog a közös használatra kiadott erőforrásért. A harmadik entitás számára elég másolni az előző feladat során létrehozott zárt rendszert, amelyben a futási idő alatt tevékenykedni fog. A modellben ezúttal az entitások az alábbi paramétereket kapták:

Első tanuló paraméterei

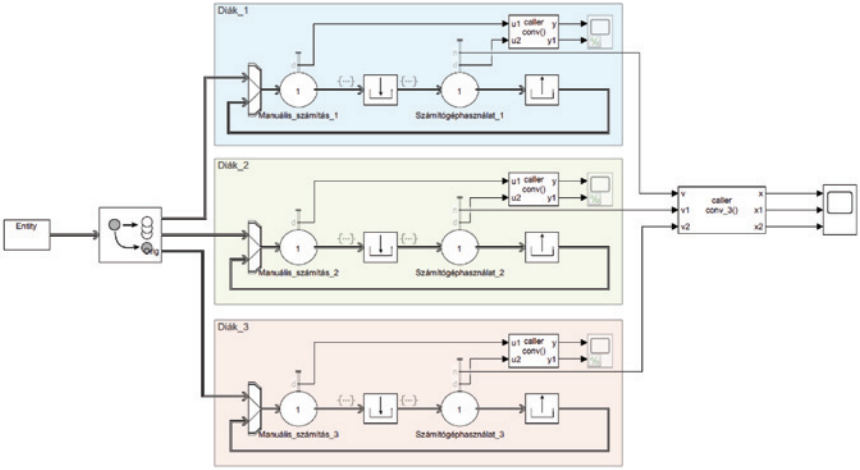
- Manuális számítás: $m = 5, M = 15$
- Számítógéphasználat: $m = 2, M = 5$

Második tanuló paraméterei

- Manuális számítás: $m = 2, M = 5$
- Számítógéphasználat: $m = 5, M = 15$

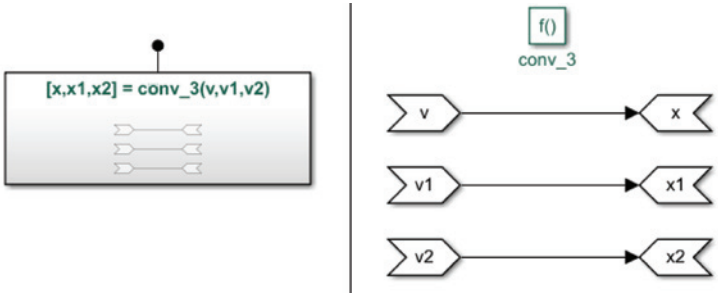
Harmadik tanuló paraméterei

- Manuális számítás: $m = 3, M = 8$
- Számítógéphasználat: $m = 3, M = 8$



112. Ábra – Erőforrás használatának megfigyelése 3 entitás esetén

Ezek után már csak a Scope több bemenetű használatához szükséges függvényt kell megadni. Mivel 3 entitással dolgozik a rendszer, ezért a függvénynek is 3 változó értékét szükséges kezelni: *[x, x1, x2] = conv_3(v, v1, v2)* (lásd 113. Ábra).

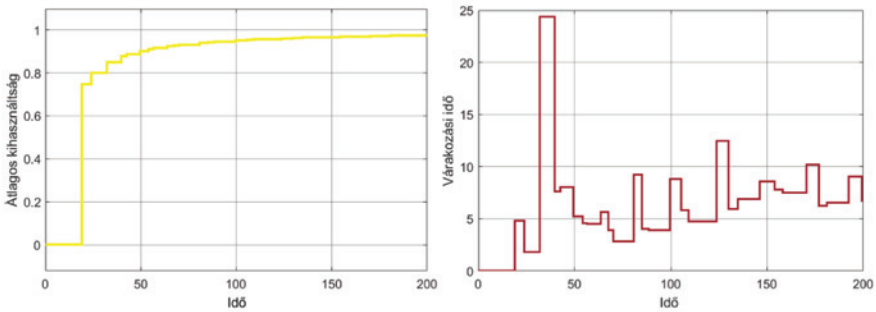


113. Ábra – 3 bemenettel és 3 kimenettel rendelkező függvény

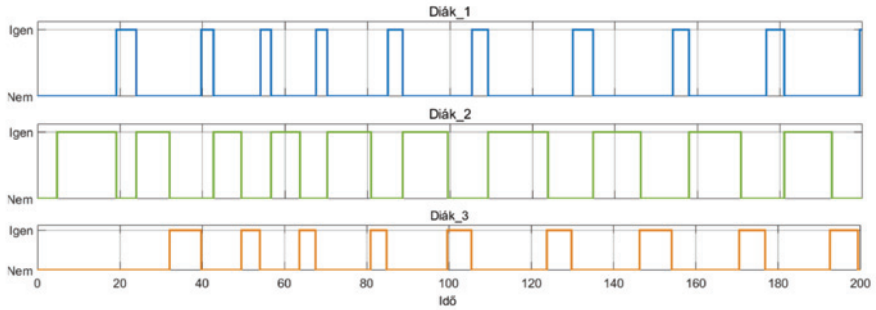
A függvényhez alkalmazkodva elérhetővé válik a **Function Caller** 3 bemenettel és 3 kimenettel rendelkező változata, ami segítségével a vizualizációs blokk már képes lesz kezelni az adatokat. A rendszer futási ideje ezúttal is 200 időegység volt, ami a megadott paraméterek mellett $y = 6.74453$ egységnyi

átlagos várakozási időt jelentett a rendszer 3 entitása számára. A számítógép kihasználtsága 97.6% volt.

Az elkészített diagramok betekintést engednek a rendszer részletes működésébe és közös erőforrásuk kezelésébe. A 114. Ábrán leolvasható, hogyan normalizálódott a kihasználtság és érte el 200 időegység alatt a util = 0.976 értéket. A 115. Ábrán a 3 szín a 3 entitást jelöli, míg 0 érték arra utal, hogy adott egyed nem használt számítógépet, míg 1 érték esetén igen. Az ábrából kiolvasható, hogy az erőforrást először a 2 diák kezdte el használni megközelítőleg az 5. időegységnél (percnél).



114. Ábra – Átlagos kihasználtság értékének (sárga) és várakozási idő alakulása az idő teltével



115. Ábra – Erőforrás használatának megoszlása az entitások között

5. HIBRID RENDSZEREK (DEV&DESS)

A hibrid rendszerek és modellek jelentős szerepet játszanak a modellezés és szimuláció területén, különösen olyan összetett rendszerek esetében, amelyek egyszerre tartalmaznak folytonos (kontinuus) és diszkrét dinamikát. E rendszerek lényege, hogy egyaránt kezelik a diszkrét események (pl. kapcsolók, döntési pontok) és a folytonos folyamatok (pl. fizikai mozgások, hőmérséklet-változások) által okozott változásokat. A hibrid rendszerek tehát olyan jelenségeket írnak le, ahol a rendszer viselkedése nem írható le kizárólag egyetlen típusú dinamikával. Ezen kombinált rendszerek gyakran jelennek meg a modern technológiai alkalmazásokban, mint például az autonóm járművek, az intelligens közlekedési rendszerek, a robotika és az ipari automatizálás területén. Ezekben az alkalmazásokban a rendszer viselkedését folytonos folyamatok és diszkrét események kombinációja határozza meg. Például egy autonóm jármű esetében a mozgás dinamikáját folytonos differenciálegyenletek írják le, míg a közlekedési szabályok és a forgalmi helyzetek diszkrét eseményekkel reprezentálhatók.

A hibrid rendszerek modellezése olyan matematikai modellek létrehozását igényli, amelyek képesek ezen komplex dinamikák pontos leírására és előrejelzésére. A hibrid modellezési lehetőségek közé tartoznak például a hibrid automaták, ahol az állapotok közötti átmenetek diszkrét események által vezéreltek, és az egyes állapotokban a dinamikát differenciálegyenletek írják le. Emellett léteznek hibrid Petri-hálók, amelyek folytonos és diszkrét komponenseket egyaránt tartalmaznak, lehetővé téve a rendszerek strukturált és részletes modellezését. Ezen módszerek részletesebb ismertetését később, az 5.2-es fejezet taglalja. A következőkben vezessük be a hibrid modellek leírására szolgáló DEV&DESS *(Discrete Event & Differential Equation System Specification)* rendszereket.

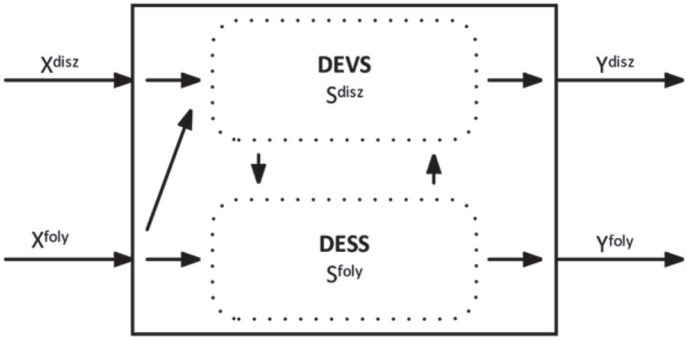
A DEV&DESS egy struktúra (Zeigler, Muzy, & Kofman, 2019):

$$DEV\ \&\ DESS = (X^{discr}, X^{cont}, Y^{discr}, Y^{cont}, S, \delta_{ext}, C_{int}, \delta_{int}, \lambda^{discr}, f, \lambda^{cont})$$

ahol:

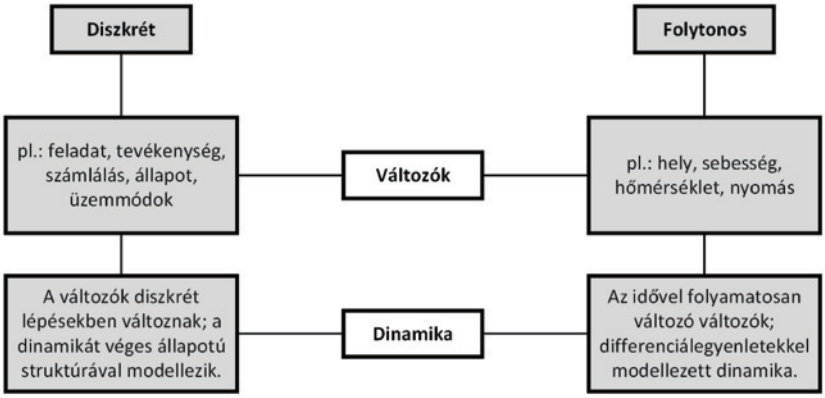
- X^{discr}, Y^{discr} a modell eseménybemeneteinek és kimeneteinek halmaza,
- $X^{cont} = \{(x_1^{cont}, x_2^{cont}, ...) | x_1^{cont} \in X_1^{cont}, x_2^{cont} \in X_2^{cont}, ...\}$ folytonos bemeneti értékek strukturált halmaza x_i^{cont} bemeneti változóval.
- $Y^{cont} = \{(y_1^{cont}, y_2^{cont}, ...) | y_1^{cont} \in Y_1^{cont}, y_2^{cont} \in Y_2^{cont}, ...\}$ folytonos kimeneti értékek strukturált halmaza y_i^{cont} kimeneti változóval.
- $S = S^{discr} \times S^{cont}$ diszkrét és folytonos állapotok szorzatának halmaza
- $\delta_{ext} : Q \times X^{cont} \times X^{discr} \rightarrow S$ a külső állapotátmeneti függvény, ahol $Q = \{(s^{discr}, s^{cont}, e) | s^{discr} \in S^{discr}, s^{cont} \in S^{cont}, e \in \mathbb{R}_0^+\}$ az összes állapot halmaza
- $\delta_{int} : Q \times X^{cont} \rightarrow S$ a belső állapotátmenet függvény
- $\lambda^{discr} : Q \times X^{cont} \rightarrow Y^{discr}$ az esemény kimeneti függvény
- $\lambda^{cont} : Q \times X^{cont} \rightarrow Y^{cont}$ a folyamatos kimeneti függvény a folyamatos kimeneti változó értékeinek meghatározásához
- $f : Q \times X^{cont} \rightarrow S^{cont}$ a változás sebességének függvénye
- $C_{int} : Q \times X^{cont} \rightarrow$ Bool az állapotesemény feltételfüggvénye az állapotesemények végrehajtásának kondicionálására.

A *DEV & DESS* vagy hibrid rendszerek egy olyan modellezési koncepció, amelyben a DEVS és a DESS elemek is megtalálhatóak. Az X^{discr} bemeneti portjai eseményszegmenseket, az X^{cont} bemeneti portjai pedig darabonként folytonos, vagy darabonként konstans szegmenseket fogadnak el. Ez utóbbi mindkét modellrészt, míg az eseménybemenet csak a DEVS részt befolyásolja. Minden rész saját, megfelelő kimenetet állít elő, mint Y^{discr} és Y^{cont}. A részek egymás állapotát is képesek befolyásolni (Kmeť, Modellezés és Szimuláció 2020, 2018) (Kmeť, Modellezés és szimuláció elmélete és eszközei, 2021).



116. Ábra – Hibrid rendszerek felépítése

A kombinált (vagy hibrid) rendszer egy olyan dinamikus rendszer, amely folytonos és diszkrét dinamikus viselkedést is mutat (Lásd 117. Ábra).



117. Ábra – Hibrid rendszerek felépítése

A modellezni kívánt rendszereket több szempont alapján is osztályozni tudjuk, mint például felépítésük, jellegük, vagy akár a vizsgált időtartomány szerint (Chaturvedi, 2017). Az időbeli változás vizsgálatának tekintetében már beszéltünk diszkrét és folytonos modellekről. Jelen fejezet keretein belül, a hibrid rendszerekre fektetjük a hangsúlyt. Ezen rendszerek jellemzően azért jelennek meg a számítógépes vezérlésben, hogy lehetővé tegyék az összetett fizikai rendszerek kényelmes modellezését. Mint ahogyan azt a nevük is sugallja, a hibrid rendszerek a folytonos és a diszkrét dinamikus rendszerek viselkedésének kombinációjának tekinthetők. A folytonos és a diszkrét szempontok két különböző világnak felelnek meg, amelyek két különböző nézetet adnak az adott rendszerről. Ez a két nézőpont együttesen egy környezetet alkot. Mivel ez a két dinamika egymás mellett létezik és kölcsönhatásban van egymással, fontos olyan modelleket kidolgozni, amelyek képesek tükrözni ezt az együttműködést. A hibrid rendszereknek ebben (is) rejlik a lényeges feladatuk. Ezidáig már számos modellt kifejlesztettek az ilyen rendszerek leírására: pl. hibrid automaták, hibrid állapotábrák vagy a hibrid Petri-hálók, amelyek működését jelen egyetemi jegyzet keretein belül is megvizsgálunk (Alur, és mtsai., 1995).

Mivel a hibrid modellek felépítésüknél fogva tartalmaznak vegyes elemeket, így a grafikus kimeneteik is különbözők. A hibrid szimulációt akkor alkalmazzák, amikor a változók egy része folytonos, míg a többi diszkrét. Ez két almodell összekapcsolásán és kölcsönhatásán alapul. A folytonos és diszkrét szimulációk váltakozva haladnak előre. A folytonos szimuláció a rendszer folytonos dinamikájával foglalkozik, és akkor hajtják végre, amikor nem észlelnek eseményt.

Az esemény lehet előre látható (például egy időbeli esemény) vagy előre nem látható (például amikor a modell egyes változói meghaladnak egy küszöbértéket). Amikor egy esemény bekövetkezik, a numerikus algoritmus átadja a vezérlést a diszkrét résznek, és várakozik, amíg stabil állapotot nem érnek el (Sahbani & Pascal, 2000). A két különböző aspektus együttes jelenléte számos kihívást okoz a szimuláció során. Valójában a fő probléma a két modell közötti szinkronizáció biztosítása. Maga a szimuláció az ilyen modellek validálásának egyik módszere.

A hibrid rendszerek tervezése és elemzése nehezebb, mint a csak diszkrét, vagy csak folytonos rendszerek tervezése és elemzése, mivel a diszkrét dinamika hatással van a folytonos fejlődésre és fordítva. A diszkrét és folytonos viselkedés közötti kapcsolatok többnyire nagyon szorosak, ezért a diszkrét-folytonos kapcsolatok modellezésében gyakori, hogy a diszkrét eseményeket a folytonos dinamika pillanatnyi változásaként ábrázoljuk. Emiatt a legtöbb gyakorlati esetben a diszkrét és folytonos dinamikai természetű rendszerek szabályozási sémáinak szintézisét még mindig heurisztikus szabályokkal közelítik meg, általában mérnöki meglátások és tapasztalatok alapján, de ez a megközelítés következésképpen hosszú tervezési és verifikációs időt igényel (Drogňa, 2012). Az irányítási közösség érdeklődését több, az iparban egyértelműen látható tendencia motiválja, amelyek új eszközök létrehozását igénylik a hibrid rendszerek irányítási sémáinak tervezéséhez, valamint stabilitásuk, biztonságuk és teljesítményük elemzéséhez. Ezen igények alapján jelenleg, számos problémát vizsgálnak a hibrid rendszerek elméletében. Itt megemlíthetjük példaként a pályák meghatározását és számítását, stabilitás- és biztonságelemzést, szabályozást, állapotbecslést stb. (Alberto, 2003).

Összesítve elmondható, hogy a hibrid rendszer előnye, hogy a rendszerek nagyobb osztályát foglalja magában, és nagyobb rugalmasságot tesz lehetővé a folytonos és diszkrét dinamikus jelenségek modellezésében. Jobban alkalmasak a nagyméretű rendszerek modellezésére, amelyek jellemzően a többágensű rendszerekben keletkeznek, hogy stabil belső dinamikával rendelkezzenek, bár a hibrid modellek összetettségük miatt nagy számítási teljesítményt igényelnek. Fontos kiemelni, hogy a hibrid rendszerek modellezése olyan matematikai problémákat hoz létre, amelyek az úgynevezett NP-nehéz problémák csoportjába tartoznak, ami azt jelenti, hogy a számítási idő a probléma dimenziójával (a változók számával) a legrosszabb esetben exponenciálisan nőhet (Drogňa, 2012). Az ilyen típusú problémák kezeléséhez elegendő számítási eszközök viszont, már elérhetők, így a hibrid rendszerek feltárásának és alkalmazásának széleskörű lehetőségei adottak a kutatók számára. Ezért a hibrid rendszerek jelenleg nagyon

népszerű és fontos kutatási területté váltak mind a tudományos, mind az ipari kutatók körében (Beek, és mtsai., 2003).

5.1. A hibrid rendszerek felhasználásának lehetőségei

A most következő fejezetünkben, azon alkalmazási területeken előforduló rendszerek működését tekintjük át, amelyek tartalmaznak hibrid rendszereket. Drogňa *Efficient Modeling of Hybrid Systems* című munkája a következőképpen osztja fel és jellemzi a hibrid rendszereket (Drogňa, 2012):

- **Mechanikus rendszerek *(Mechanical systems)***: Ezekben a rendszerekben a folyamatos mozgást, ütközések szakíthatják meg, vagy különböző üzemmódokban működhetnek, ami a véges állapotú gépek diszkrét eseményeiként írhatók le. Ilyen rendszer lehet például egy sebességtartó rendszer, amely a sebességváltó sebességfokozatát (diszkrét bemenet), a motor nyomatékát (folytonos bemenet) és a fékerőt (folytonos bemenet) vezérli annak érdekében, hogy a jármű a kívánt sebességét kövesse, miközben minimalizálja az üzemanyag-fogyasztást és a károsanyag-kibocsátást (Alberto, 2003). Egy másik példa a benzinmotor működése, amely a mechanikus rendszereken belüli hibrid viselkedést mutatja be. Ennél a példánál a hajtáslánc, a gázáramlás és a termikus dinamika folytonos folyamatoknak számítanak, míg a dugattyúk négy diszkrét működési móddal rendelkeznek (Alberto, 2003).
- **Elektromos áramkörök *(Electrical circuits)***: Itt a folytonos jelenségeket – például a kondenzátorok töltése – a kapcsolók nyitása és zárása, vagy akár a diódák be- és kikapcsolása szakíthatja meg. Későbbi példáinknál, egy ilyen hibrid rendszer működésére is kitérünk.
- **Kémiai folyamatszabályozás *(Chemical process control)***: A kémiai folytonos reakciókat diszkrét műveletekkel szabályozzák – például szelepek és szivattyúk nyitogatásával. További részletes leírást a hibrid rendszerek vegyipari szerepéről Ashish *Logical Modeling Frameworks for the Optimization of Discrete-Continuous Systems* című disszertációjában olvashatunk. (Ashish, 2006).
- **Beágyazott számítási rendszerek *(Embedded computation systems)***: Hibrid modelleket alkalmazunk a beágyazott rendszerek azon eseteiben, amikor a digitális számítógép interakcióba lép egy többnyire analóg környezettel. A beágyazott rendszer olyan számítógépes rendszer, amelyet meghatározott feladatok elvégzésére terveztek, általában egy nagyobb rendszer részeként. Ilyen rendszerekről beszélhetünk a közúti járművek, illetve a repülőgépek kapcsán.

- **Hálózati vezérlőrendszerek *(Networked control systems)***: A hibrid rendszerek fontos osztálya, ahol az érzékelés, vezérlés és működtetés nem közvetlenül, hanem egy közös hálózati médiumon keresztül kapcsolódnak egymáshoz (Zhang, Branicky, & Philips, 2001).
- **Komplex rendszerek *(Complex systems)***: Hierarchikusan szerveződnek, ahol például a magasabb szintű diszkrét tervezési algoritmusok kölcsönhatásba lépnek az alacsonyabb szintű folyamatos irányítási algoritmusokkal és folyamatokkal (Alberto, 2003).
- **Többmodell-rendszer *(Multiple model systems)***: Ezek olyan rendszerek, amelyek általános modelljét és azok általános fejlődését különböző almodellek szabályozzák. Ez a szabályozás lehet az állapottér felosztása a hozzárendelt almodellekkel rendelkező régiókra (pl. darabonkénti affin rendszerek). Egy másik szabályozási lehetőség a rendszer paramétereinek változtatása az adott jelnek megfelelően (pl. kapcsolt rendszerek vagy üzemmódváltásos rendszerek). Ezen modellek alkalmazásai a mérnöki gyakorlatban például a repülésirányítási és légiforgalmi irányítási rendszerekben jelennek meg (Lygeros, Sastr, & Tomlin, 2012).
- **Adaptív rendszerek *(Adaptive systems)***: Az adaptációs törvényt, például darabonkénti affin rendszerek, vagy véges állapotú gépek által meghatározott kapcsolási szabályok biztosítják.
- **Modellezett hibákkal rendelkező rendszerek *(Systems with modeled failures)***: A rendszerben bekövetkező meghibásodás vagy hirtelen fellépő hibák esetén, a hiba előfordulása kapcsolójelként modellezhető. A hibás rendszer ekkor hibrid rendszernek tekinthető (Drogňa, 2012).

5.2. A hibrid rendszerek modellezésének lehetőségei

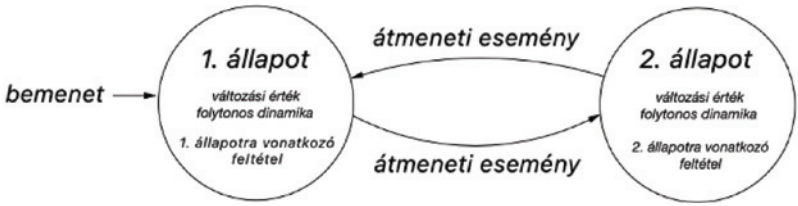
Még mielőtt komplexebb példákon mutatnánk be a hibrid rendszerek modelljeit, és vizsgálnánk meg azok kimeneteit, tekintsük meg a hibrid modelltípusokat az alábbi jegyzetek alapján:

- Sahbani *Simulation of Hybrid Systems Using Stateflow* című munkája a hibrid rendszerek szimulációjával foglalkozik, melyben olyan hibrid modellezési eljárásokat mutat be, mint a hibrid automaták, hibrid állapotábrák és hibrid Petri-hálók (Sahbani & Pascal, 2000). (Bail, Alla, & David, 1991; Harel, 1987; Harel, 1987) Harel *Statecharts: A Visual Formalism for Complex Systems* című munkája az állapotábrák felhasználási lehetőségeit mutatja be komplex rendszerek modellezésére (Harel, 1987).

- Alur és társai *The algorithmic analysis of hybrid systems* című munkája a hibrid automaták működésére összpontosít. Tanulmányuk a hibrid rendszereket véges automatákkal modellezi (Alur, és mtsai., 1995).
- Bail *Hybrid Petri Nets* című munkája (Bail, Alla, & David, 1991).

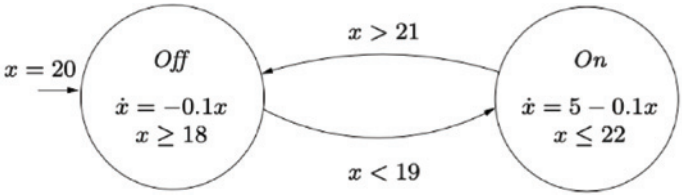
Hibrid automata

A hibrid automaták valós értékű változók véges halmazával dúsított, véges állapotú automatákként értelmezhetők (Alur, és mtsai., 1995). Egy adott pontban (diszkrét állapotban) a változók értékei folyamatosan változnak az időhöz kapcsolódó differenciálegyenletek szerint. Ez mindaddig elmondható, amíg az adott pont invariáns. Amikor az átmeneti feltétel (a folytonos változókra vonatkozó feltétel(ek)) teljesül, a rendszer egy másik állapotba kerül. Az alábbiakban tekintsük át az ilyen modellek általános felépítését.



118. Ábra – Hibrid automata felépítése

Most pedig egy konkrét példán demonstráljuk egy hibrid automata működését Beek és társai *Relating Chi to Hybrid Automata* című munkájában bemutatott termosztát modell alapján (Beek, és mtsai., 2003) termosztát modellje alapján.



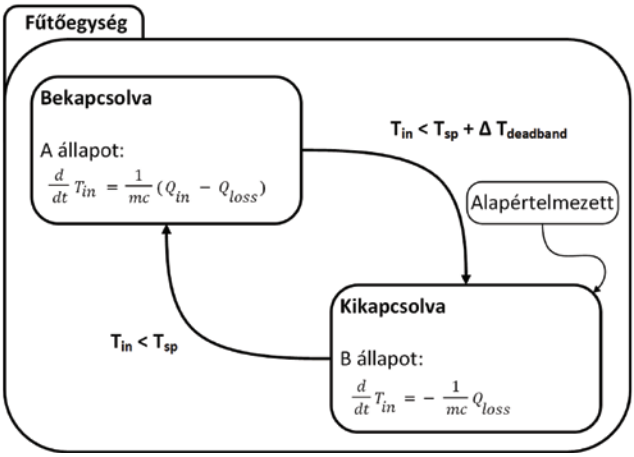
119. Ábra – Termosztát modell felépítése hibrid automata segítségével

Az ábrából kiolvasható, hogy a kezdeti hőmérséklet 20 fok ($x = 20$), és a fűtés ki van kapcsolva (kikapcsolt vezérlési mód → 1. állapot). A hőmérséklet az áramlási feltételnek megfelelően csökken $\dot{x} = -0.1x$. Az $x < 19$ átmeneti feltétel szerint a fűtőberendezés bekapcsolhat, amint a hőmérséklet 19 fok alá

csökken. Az $x \geq 18$ invariáns feltétel miatt a fűtőberendezés legkésőbb akkor kapcsol be, amikor a hőmérséklet 18 fokos. A bekapcsolt szabályozási módban (2. állapot) a fűtőberendezés bekapcsol, és a hőmérséklet az $\dot{x} = 5 - 0.1x$ folytonos dinamikának megfelelően emelkedik. Amikor a hőmérséklet 21 fok fölé emelkedik, a fűtőberendezés kikapcsolhat. Az $x \leq 22$ invariáns feltétel miatt, a fűtőberendezés legkésőbb akkor kapcsol ki, amikor a hőmérséklet eléri a 22 fokot.

Hibrid állapotábra

Az állapotábrák formalizmusát Harel Statecharts: *A Visual Formalism for Complex Systems* című munkája taglalja, ahol az állapotábrákat egy olyan jelöléssel egészítik ki, amely lehetővé teszi, hogy egy alapállapotot egy differenciálegyenlettel írhassunk le. Ennek az az implikált jelentése, hogy amikor az állapot aktív, akkor a hozzá tartozó differenciálegyenlet működésbe lép (Harel, 1987). Az átmenetek tipikusan esemény és az arra reagáló folyamat – mondhatni akció/reakció – alakúak. Az esemény váltja ki az átmenetet, a választható művelet pedig az átmenet bekövetkezésekor kerül végrehajtásra. Ezenkívül egy esemény, kapcsolódhat egy folytonos változóhoz is, például egy küszöbérték átlépésekor. Az eseményre reagáló folyamat alatt pedig, a folytonos változó frissítését is érthetjük (Sahbani & Pascal, 2000). A következőkben tekintsük meg a hibrid állapotábra alkalmazását egy fűtési rendszeren keresztül Yahiaoui *Simulation based Design Environment for Multi-Agent Systems in Buildings* című munkája alapján (Yahiaoui, Hensen, Soethout, & Paassen, 2006).



120. Ábra – Fűtési rendszer felépítése hibrid állapotábra segítségével

A 120. Ábra egy épület fűtési rendszerét mutatja be. Ez a fűtési rendszer két változóval rendelkezik:

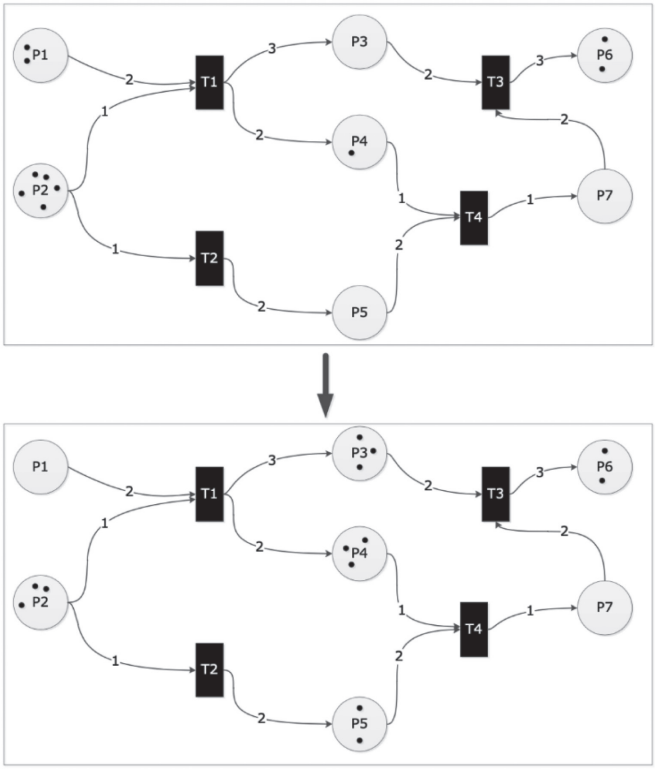
- T_{sp} a beállított hőmérséklet
- T_{in} a belső levegő hőmérséklete

Ezek mellett kiolvasható két diszkrét állapot A és B, amelyek a fűtési rendszer működési módjára utalnak. Mindkét állapotot differenciálegyenletek írják le. Az ábrán látható harmadik változót – $T_{deadband}$ elhanyagolható hőmérsékletű változó – alapvetően a mozgáshoz használják, amely a rendszer belső paramétereit reprezentálja a valós idejű specifikációhoz (pl. a szabályozó érzékenysége, a szabályozott változó válaszának késleltetése stb.). A rendszer alapértelmezés szerint a B állapotból indul, ahol a fűtőberendezés ki van kapcsolva. A B állapotból az A állapotba való átmenet akkor történik meg, ha a belső hőmérséklet T_{in} értéke, a referenciaérték T_{sp} alá csökken. Amikor ez az átmenet megtörténik, a belső fűtési sebesség értéke azonnal növekedésnek indul. A változó hőáramban 0% (amikor a fűtőberendezés teljesen ki van kapcsolva) és 100% (amikor a fűtőberendezés teljesen be van kapcsolva) A B állapotba való visszalépés akkor történik meg, amikor a belső hőmérséklet (T_{in}) a beállított érték (T_{sp}) fölé kerül.

Hibrid Petri-hálók

A hibrid Petri-hálók jobb megértése érdekében, röviden tekintsük meg, hogy mit értünk Petri-háló alatt. Ez a modell a Place/Transition Petri NETS és a differenciálalgebrai egyenletek kombinációja. A Petri-hálókat, a rendszer különböző konfigurációinak, azaz diszkrét szempontjainak ábrázolására használják (Bail, Alla, & David, 1991). A Petri-háló egy irányított, kétszínű és kétrészes gráf. A kétszínű és kétrészes tulajdonsága azt jelenti, hogy két egyedi csomóponthalmazra oszlik, amelyeket *átmeneteknek (tranzíció)* és *helyeknek* nevezünk, és a kétrészes tulajdonság szerint csak a *helyek* kapcsolódhatnak *átmenetekhez* vagy *átmenetek a helyekhez*. A *helyeket* grafikusan körökkel, az *átmeneteket* pedig téglalapokkal ábrázoljuk. Az adott *helyekben* ún. *tokenek* (jele fekete pötty, vagy szám) helyezkedhetnek el. A *tokenek* egy sor információt hordoznak, a következő műveleti differenciálegyenletekben használt, különböző paraméterekre és főként egyes változók frissítésére vonatkozóan. A *tokenek* kiválasztását, az *átmenetek* kiváltására vonatkozó, további feltételek határozzák meg. A következőkben bemutatjuk a Petri-hálók működési elvét, majd egy konkrét példán szemléltetjük a hibrid Petri-hálókat.

Minden *hely* egész számú *tokent* tartalmazhat. Ezek a *tokenek* lehetnek a már említett grafikus fekete pontok vagy számok a *helyek* belsejében. Egy *hely token*-számának konkrét meghatározását a *hely* állapotának nevezzük, és minden *hely token*-számának konkrét meghatározását a Petri-háló állapotának nevezzük. Egy *átmenet* akkor tudja ezeket a *tokeneket* kilőni, ha az előző területének (előző *helyeknek*) minden *helye* legalább annyi *tokennel* rendelkezik, mint az adott *él* súlyozása. Az *élek* irányított nyilak, amelyek a *helyeket* az *átmenetekkel* és az *átmeneteket* a *helyekkel* kötik össze. A Petri-hálóban minden *helynek* lehet egy alsó és egy felső *token* határa. Ezeket a Petri-hálókat kapacitással rendelkező Petri-hálóknak nevezzük. Továbbiakban fontos tisztázni, hogy mit értünk tüzelésre kész *átmenet* alatt. Egy tüzelésre kész *átmenet* úgy tüzel, hogy annyi *tokent* távolít el az összes korábbi helyéről, amennyit a megfelelő élsúlyozás meghatároz, és annyi *tokent* ad hozzá az összes korábbi területének (korábbi *helyek*) összes *helyéhez*, amennyit az élsúlyozás meghatároz.



121. Ábra – Példa Petri-háló felépítésére és átmenetére (Proß & Bachmann, 2011)

A 121. Ábra felül egy Petri-háló példáját mutatja, ahol az 1. és 2. *átmenet* tüzelésre kész, a többi nem. Az alsó Petri-háló az 1-es és 2-es tranzíció elsütése utáni, új állapotot mutatja.

További leírást és egyszerű példákat a Petri-hálókról Proß és Bachmann *An Advanced Environment for Hybrid Modeling of Biological Systems Based on Modelica* című munkája, valamint Bartha, Majzik és Pataricza *Petri hálók: Alapelemek és kiterjesztések* című egyetemi jegyzete mutat be (Bartha, Majzik, & Pataricza, 2021) (Proß & Bachmann, 2011). Most pedig tekintsük meg, hogy mit értünk hibrid Petri-háló alatt.

Hibrid Petri-hálónak nevezzük azokat a hálókat, amelyek diszkrét, és folytonos Petri-háló elemeket is tartalmaznak. Felírható egy olyan (*DH, FH, DA, FA, F, G, fd, fc, k,* M_0) halmaz formájában, ahol:

- $DH = \{DH_1, DH_2, ..., DH_{dh}\}$ – *diszkrét helyek véges halmaza,*
- $FH = \{FH_1, FH_2, ..., FH_{fh}\}$ – *folytonos helyek véges halmaza,*
- $DA = \{DA_1, DA_2, ..., DA_{da}\}$ – *diszkrét átmenetek véges halmaza,*
- $FA = \{FA_1, FA_2, ..., FA_{fa}\}$ – *folytonos átmenetek véges halmaza,*

ahol érvényesek az alábbi tulajdonságok:

$DH \cap DA = \emptyset$, $FH \cap FA = \emptyset$, $DH \cap FH = \emptyset$, $DA \cap FA = \emptyset$, $DH \cap FA = \emptyset$ és $FH \cap DA = \emptyset$.

Továbbá,

$F \subseteq (DH \times DA) \cup (FH \times FA) \cup (FH \times DA)$ élek halmaza,
$G \subseteq (DA \times DH) \cup (FA \times FH) \cup (DA \times FH)$ élek halmaza

$fd:(F \cup G)\backslash\{(FH \times FA) \cup (FA \times FH)\} \rightarrow \varsigma$ jelöli az élek súlyozási függvényét, amely minden élhez hozzárendel egy $g \in \varsigma : \mathbb{N}^{dh} \rightarrow \mathbb{N}$ függvényt, a diszkrét helyek állapotának egy részhalmazától függően.

$fc:(F \cup G)\backslash\{(DH \times FA) \cup (DA \times DH) \cup (FH \times DA) \cup (DA \times FH)\} \rightarrow H$ jelöli az élek súlyozási függvényét, amely minden élhez hozzárendel egy $h \in H$: $\mathbb{R}_+^{dh+fh} \rightarrow \mathbb{R}_+$ függvényt a diszkrét, és folytonos helyek állapotának egy részhalmazától függően.

$k:DA \rightarrow \mathbb{R}_+^{da}$ jelöli a késleltetési függvényt
$M_0:DH \rightarrow \mathbb{N}_+^{dh}, FH \rightarrow \mathbb{R}_+^{fh}$ jelöli a kezdeti állapotot.

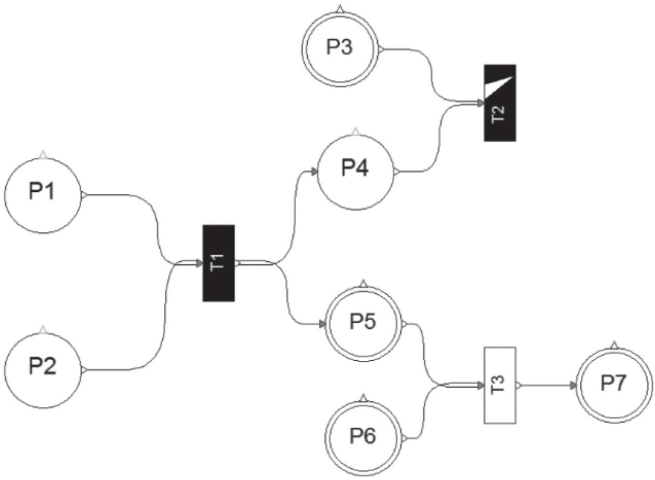
Most, hogy bemutattuk, miként tartalmaz egy hibrid Petri-háló diszkrét és folytonos elemeket egyaránt, tekintsük meg, hogy milyen kapcsolatok állhatnak fenn (megengedettek) az ilyen hálókon belül:

- diszkrét hely → diszkrét átmenet
- diszkrét átmenet → diszkrét hely
- folyamatos átmenet → folyamatos hely
- folyamatos hely → diszkrét átmenet
- diszkrét átmenet → folyamatos hely

Nem engedélyezett viszont, az alábbi két kapcsolat:

- folytonos hely → folytonos átmenet
- folytonos átmenet → diszkrét hely

A 122. Ábra szemlélteti a hibrid Petri-háló felépítését (Proß & Bachmann, 2011).



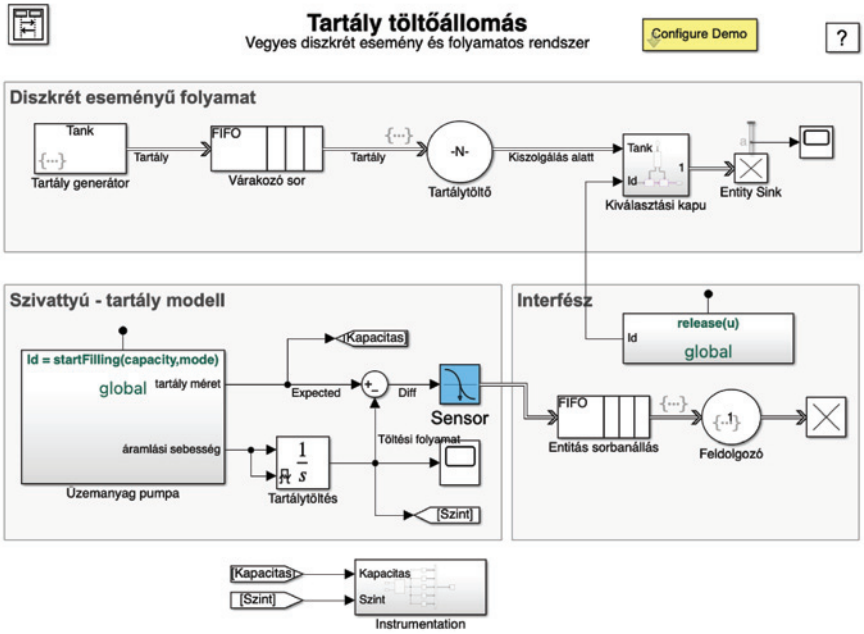
122. Ábra – Hibrid Petri-háló felépítése, ahol P1, P2, P4 és T1 diszkrét, T2 sztochasztikus és P3, P5, P6, P7 és T3 folytonos Petri-háló elemek.

5.3. Példák hibrid rendszerekre

A modellek blokkdiagramokból álló felépítését tekintve, a hibrid rendszerek tartalmaznak diszkrét és folytonos komponenseket egyaránt. Az ilyen rendszerek általában valamilyen folytonos fizikai folyamatból állnak, amelyet diszkrét logikai komponensek vezérelnek (Dabney & Harman, 2004). Következő példáinkon (MathWorks, The MathWorks, Inc., 2021) tekintsünk át az ilyen rendszerek felépítését és működését.

5.3.a. Példa - Tartálytöltés

Ez a példa egy olyan hibrid rendszert mutat be, amely folytonos idejű és diszkrét eseményű szakaszokat is tartalmaz. A diszkrét eseményű rész a tartályokat modellezi, amelyek sorban állnak, és amelyeket fel kell tölteni valamilyen folyadékkal. Minden tartály rendelkezik bizonyos nagyságú kapacitással, amely korlátozza a túlcsordulást. A folytonos idejű szakasz a tartály feltöltésének folyamatát modellezi. Amikor egy tartály eléri a meghatározott kapacitást, tehát megtelik, akkor a rendszer ezt egy eseményként érzékeli. Ezt követően a rendszer, egy az eseménynek megfelelő üzenetet generál, amely a későbbiekben az érintett tartály felszabadítását eredményezi. Az alábbiakban tekintsük meg ezen modell felépítését, és a modellt alkotó komponensek funkcióit.



123. Ábra – Tartálytöltés modell felépítése (MathWorks, The MathWorks, Inc., 2021)

Az egyes blokkok funkcióit az alábbiakban ismertetjük:
Tartály generátor: Rendszeresen generál tartályokat, amelyek mindegyikéhez tetszőlegesen hozzárendelt kapacitás-attribútum tartozik.

Várakozó sor: Sorba állítja a feltöltésre váró tartályokat.

Tartálytöltő: Kiszolgálja a tartályokat, és meghívja a *startFilling* Simulink-funkciót, hogy átadja a tartály kapacitás-attribútumát a modell időalapú részének.

Tartálytöltés: Az egyes tartályok kapacitásig történő feltöltésének folyamatát modellezi.

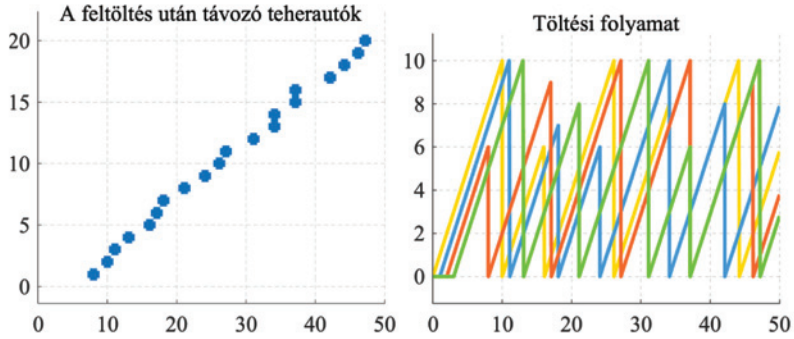
Sensor: Érzékeli, ha a tartályba töltött mennyiség elérte a kapacitást, s amikor ez megtörténik, üzenetet küld a modell diszkrét esemény alapú szakaszának. Az érzékelő, hídként szolgál az időalapú szakasz, és a páros alapú szakasz között.

Feldolgozó: Fogadja az érzékelőtől a generált üzenetet, és eldönti, hogy melyik tartályt kell felszabadítani a kiszolgálótól. Ezután meghívja a *release* nevű Simulink-funkciót, hogy egy adott tartályra vonatkozó felszabadítási üzenetet generáljon.

Kiválasztási kapu: Fogadja a felszabadítási üzenetet, és válaszul megnyitja a kaput, hogy átengedje az adott tartályt.

Configure Demo: Beállítja a benzinkút benzinszivattyúinak számát, és be-/kikapcsolja az animációt.

Scope: Feladata a grafikus kimenetek megjelenítése. Ezen modell esetében két kimenet írható le: diszkrét és folytonos (lásd 124. Ábra).



124. Ábra – A modell diszkrét (fent) és folytonos kimenete (lent)

A modell, az alábbi programkódon keresztül érhető el, és futtatható le a MATLAB-ban.

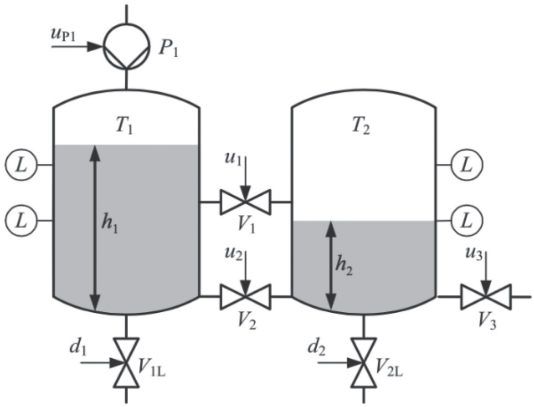
37. Forráskód – Tartálytöltés modelljének meghívása

```
% Hybrid Systems - Tank Filling

modelname = 'seTankFilling';
open_system(modelname);
scopes = find_system(modelname,'BlockType','Scope');
cellfun(@(x)close_system(x),scopes);
sim(modelname);
cellfun(@(x)open_system(x),scopes);
clear modelname
```

5.3.b. Példa - Dupla tartály modell

Itt az előző modellünket egészítjük ki két esetre. Ekkor beszélhetünk ugyanis, dupla tartály modellről, amelyet Lunze 2009-ben kiadott *Handbook of Hybrid Systems Control: Theory, Tools, Application* című könyve alapján szemléltetünk a következőkben (Lunze & Lamnabhi-Lagarrigue, 2009). Ez a hibrid rendszer autonóm kapcsolással adható meg, melynek a fő szabályozási feladata, az adott állapotnak a stabilizálása. Ezen egyszerűsített modellnek a gyakorlati alkalmazása széleskörben elterjedt. Konkrét felhasználási területe, hogy a tartályok folyadékszintjének előírt értéken tartásával, folyamatos folyadékáramlást biztosítson a fogyasztó számára.

Mindenekelőtt ismertetjük a folyamat leírását, melyet a 125. Ábra demonstrál. Ez a példa két, csövekkel összekapcsolt T_1 és T_2 hengeres tartályból áll. A tartályok közötti és a tartályokból történő vízáramlás a V_1, V_2, V_3, V_{1L} és V_{2L} szelepekkel szabályozható. Ezek csak teljesen nyitható vagy zárható (be/ki) funkciókkal ellátott szelepekként értelmezhetők. A tartályok közötti összekötő csövek a tartályok alján V_2 szeleppel, és a fenék feletti h_0 magasságban, V_1 szeleppel helyezkednek el.



125. Ábra – Dupla tartály rendszerének felépítése (Lunze & Lamnabhi-Lagarrigue, 2009)

Az egyes tartályok maximális vízszintjét h_{max} jelöli. Minden tartály azonos, A-val jelölt keresztmetszettel rendelkezik, és azonos szinten helyezkedik el.

Egy tipikus helyzetben a V_1, V_2 és V_3 szelepek nyitva vannak, a V_{1L} és V_{2L} szelepek pedig zárva. A folyadékot a P_1 szivattyú tölti a bal oldali tartályba. A mérések a T_1 és T_2 tartályokban lévő $h_1(t)$ és $h_2(t)$ szintekre vonatkoznak. A diszkrét érzékelők (az ábrán L-el jelölve) a folyadékszintek minőségi jellemzését alacsony, közepes és magas szintként adják meg.

Mivel hibrid rendszerről van szó, ezért magától értetődő, hogy folytonos és diszkrét bemenetekkel is rendelkezik. A folytonos bemenet az

$$u_{p1}(t) = Q^{p_1}(t) \tag{5.1}$$

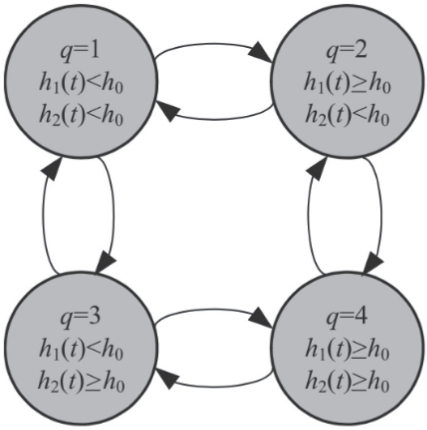
szivattyún keresztüli beáramlás. A diszkrét bemenetek pedig a V_1, V_2 és V_3 szelepek helyzete. Ebből adódóan:

$$u(t) = (u_{p1}(t)\, u_1(t)\, u_2(t)\, u_3(t))^T. \tag{5.2}$$

A rendszerre ható zavarokat a V_{1L} és V_{2L} szelepek helyzetének megváltoztatásával lehet előidézni.

A kéttartályos rendszer tipikus hibrid rendszer, mivel folyamatos dinamikával, állapotfüggő és szabályozott kapcsolással rendelkezik. Ha a szelepek helyzete

állandó marad, a folytonos dinamika autonóm módon váltogat négy $q(t)$ diszkrét üzemmód között attól függően, hogy a folyadékszintek meghaladják-e a felső csatlakozócső h_0 magasságát vagy sem. A rendszer viselkedését a már előzőkben bemutatott hibrid automata segítségével ábrázolhatjuk, ahol minden csomópont egy-egy diszkrét üzemmódot képvisel.



126. Ábra – Dupla tartály modell diszkrét állapotainak felírása hibrid automata segítségével (Lunze & Lamnabhi-Lagarrigue, 2009)

A dupla tartállyal ellátott rendszer két folytonos állapotváltozóval rendelkezik

$$x(t) = (h_1(t)\, h_2(t))^T, \quad h_i \in \mathbb{R} \tag{5.3}$$

és négy diszkrét állapottal

$$q(t) \in \{1, 2, 3, 4\} \tag{5.4}$$

amelyek a 126. Ábra automatájának állapotaiból könnyen kiolvasható, h_0, h_1 és h_2 szintek viszonyaitól függenek.

Torricelli törvényéből adódóan felírható az alábbi nemlineáris dinamika:

$$Q_{ij}^{V_l}(t) = c \cdot sgn(h_i(t) - h_j(t)) \cdot \sqrt{2g|h_i(t) - h_j(t)|} \cdot u_l(t), \tag{5.5}$$

ahol:

- $Q_{ij}^{V_l}(t)$ – jelöli a vízáramlást a T_1 tartályból a T_2 tartályba a V_l szeleppel ellátott csövön keresztül
- c – a szelepek áramlási állandóját jelöli

- $u_l\,(t) \in \{0, 1\}$ – a V_l szelep helyzete (0 azt jelenti, hogy a szelep zárva van, 1 pedig, hogy nyitva)
- g – jelöli a gravitációs állandót.

Egy tartályban lévő víz $V(t)$ térfogatának a változása a következőképpen írható fel:

$$\dot{V}(t) = \dot{h}\,(t) \cdot A = \sum Q_{in}\,(t) - \sum Q_{out}\,(t), \tag{5.6}$$

ahol:

- $\sum Q_{in}\,(t)$ – a tartályba beérkező áramlás összege,
- $\sum Q_{out}\,(t)$ – a tartályból kifolyó áramlás összege.

Ezt az egyenletet a dupla tartályra alkalmazva a következő nemlineáris differenciálegyenleteket kapjuk:

$$\begin{aligned} \dot{h}_1(t) &= \frac{u_{p_1}\,(t) - Q_{12}^{V_1}(t) - Q_{12}^{V_2}(t) - Q_L^{V_{1L}}(t)}{A}, \\ \dot{h}_2(t) &= \frac{Q_{12}^{V_1}(t) + Q_{12}^{V_2}\,(t) - Q_L^{V_{2L}}(t) - Q_N^{V_3}(t)}{A}\,. \end{aligned} \tag{5.7}$$

A $Q_{12}^{V_1}\,(t)$ áramlás értéke a következőképpen függ a $q(t)$ üzemmódtól:

$$Q_{12}^{V_1}\,(t)\begin{cases} 0, & q(t) = 1, \\ c \cdot sgn(h_1(t) - h_0\,) \cdot \sqrt{2g \cdot |h_1(t) - h_0\,|} \cdot u_1\,(t), & q(t) = 2, \\ c \cdot sgn(h_0 - h_2(t)) \cdot \sqrt{2g \cdot |h_0 - h_2\,(t)|} \cdot u_1\,(t), & q(t) = 3, \\ c \cdot sgn(h_1(t) - h_2(t)) \cdot \sqrt{2g \cdot |h_1(t) - h_2(t)|} \cdot u_1\,(t), & q(t) = 4. \end{cases} \tag{5.8}$$

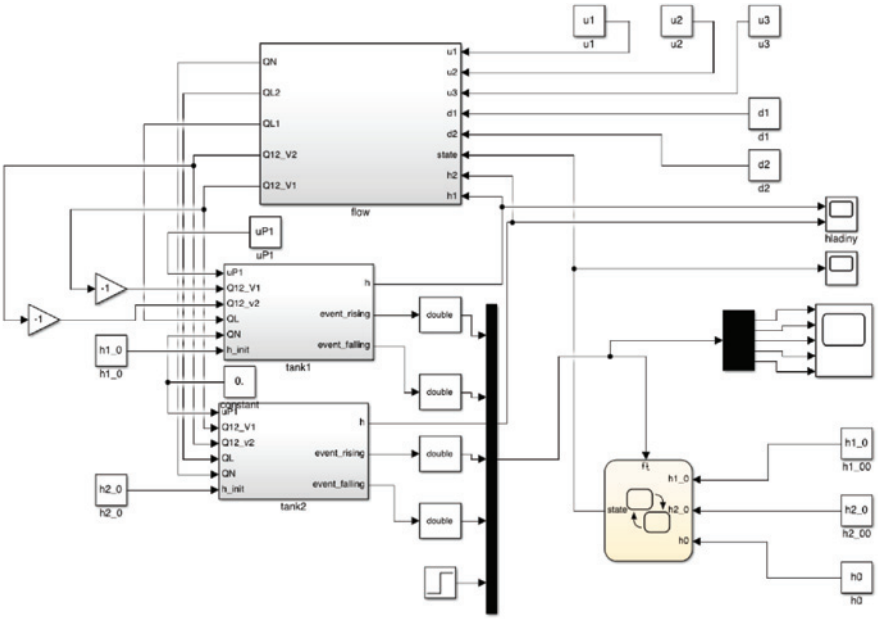
A következő egyenletek mind a négy üzemmódra érvényesek:

$$\begin{aligned} Q_{12}^{V_2}(t) &= c \cdot sgn(h_1\,(t) - h_2(t)) \cdot \sqrt{2g|h_1\,(t) - h_2\,(t)|} \cdot u_2\,(t), \\ Q_N^{V_3}(t) &= c \cdot \sqrt{2g \cdot h_2(t)} \cdot u_3\,(t), \\ Q_L^{V_{iL}} &= c \cdot \sqrt{2g \cdot h_i(t)}\ \cdot d_i(t),\ i = 1, 2, \end{aligned} \tag{5.9}$$

ahol:

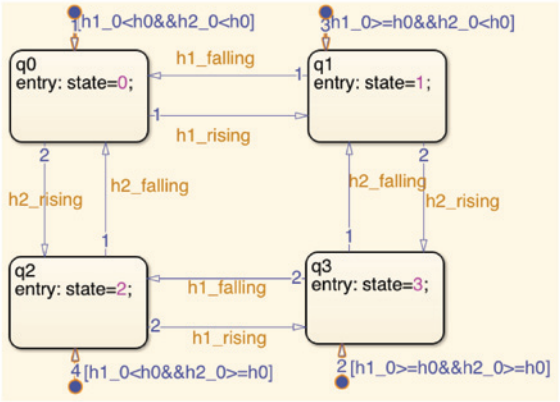
- $Q_N^{V_3}(t)$ – a T_2 tartályból a V_3 szeleppel ellátott csövön távozó vízmennyiség,
- $Q_L^{V_{iL}}$ – a T_i tartályból a V_{iL} szeleppel ellátott csövön távozó vízmennyiség.

Most pedig tekintsük meg ezen modell felépítését, valamint a hozzá tartozó paraméterek MATLAB kódját.



127. Ábra – Dupla tartály modell felépítése Simulink és Stateflow blokkok segítségével

A fentiekben felvázolt hibridautomata állapotainak digitális megvalósítását szemlélteti a 128. Ábra.



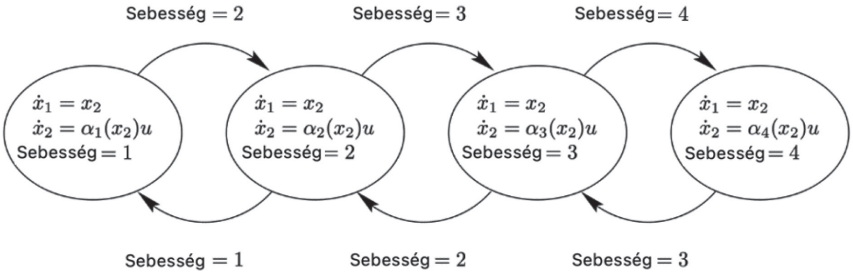
128. Ábra – A dupla tartály modell állapotainak meghatározása

38. Forráskód – Dupla tartály modell paraméterezése

```
%Two-Tank model
% setting the stateflow path
global k
A=1;B1=0.2;B2=0.2;
inflow=20;
v1=1;v2=0;v3=0;
k=1 %control of max. level
d1=0;
d2=0;
h1_0=1.1;
h2_0=1.2;
h0=1;
uP1=0;u1=1;u2=1;u3=1;
%two tank parameters
uP1=0.0003;
h1_0=.25; h2_0=.45;
h0=0.3;h_max=1;
u1=1;u2=1;u3=1;
d1=1;d2=1;
A=0.015
%tank level control
inflow=5
B1=1;
B2=1;
```

5.3.c. Példa – Sebességváltó vezérlés

A sebességváltó vezérlésének működtetése egy olyan vezérléstervezési problémát mutat be, ahol mind a folytonos, mind pedig a diszkrét vezérlést meg kell határozni. A 129. Ábra egy négyfokozatú sebességváltóval rendelkező személygépkocsi modelljét ábrázolja (Johanson, Lygeros, & Sastry, 2004).



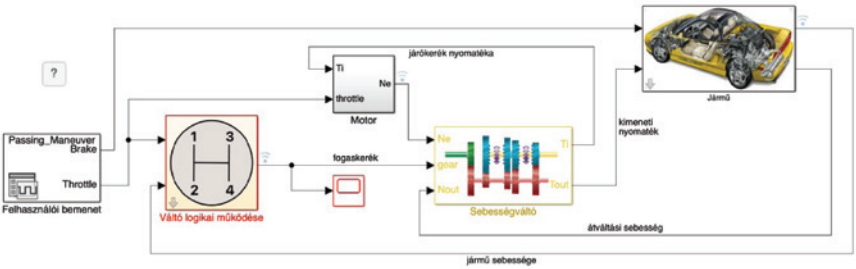
129. Ábra – Négysebességű váltó hibrid modellje

ahol:

- x_1 – jelöli a gépkocsi hosszirányú helyzetét az út mentén,
- x_2 – jelöli a gépkocsi sebességét,
- a_i – függvény jelöli az i-ik fogaskerék hatásfokát,
- *sebesség* ∈ *{1,...,4}* – jelöli a sebességváltó helyzetét,
- $u \in [u_{min}, u_{max}]$ – jelöli a gázpedál helyzetét.

Utóbbi kettő (a sebességváltó és a gázpedál helyzete) a rendszer bemenetének tekinthető, míg a pozíció és a sebesség kimenetnek. A sebességváltásra azért van szükség, mert a motor nagyon alacsony, vagy nagyon magas fordulatszámon, kevés teljesítményt tud kifejteni.

Az alábbiakban tekintsünk meg egy szintén négyfokozatú, automata sebességváltó felépítését, a már előző fejezetekben megismert Simulink és Stateflow blokkok segítségével. (MathWorks, The MathWorks, Inc., 2021)

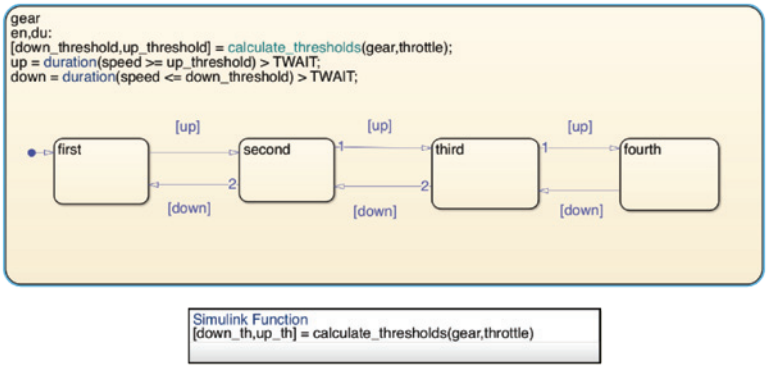


130. Ábra – Automata sebességváltó-modell felépítése (MathWorks, The MathWorks, Inc., 2021)

A modellt felépítő lényeges blokkok és azok funkciói:

- **Felhasználói bemenetek (User Inputs):** Két bemenetet biztosít a modellhez, a féket és a gázpedált.
- **Motor (Engine):** A motor fordulatszámának kiszámítása a járókerék nyomatékértéke és a gázpedál alapján.
- **Váltó logikai működése (Gear_logic):** Kiszámítja a következő sebességfokozatot az aktuális sebességfokozat, a gázpedál és a jármű aktuális sebessége alapján.
- **Sebességváltó (Transmission):** A járókerék és a kimeneti nyomaték kiszámítása a fordulatszám, a sebességfokozat és a sebességváltó sebessége alapján.
- **Jármű (Vehicle):** Kiszámítja a jármű és a sebességváltó sebességét a kimeneti nyomaték és a fék alapján.

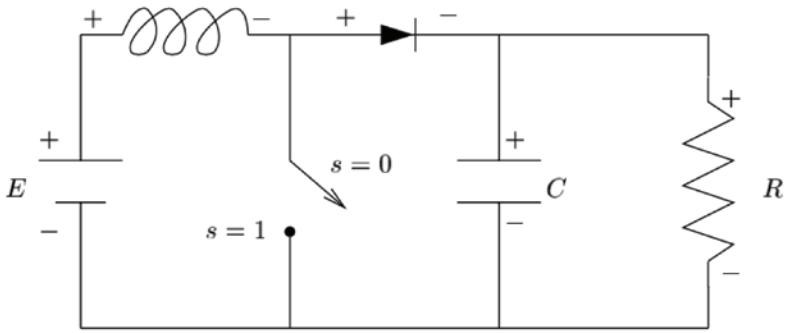
A Stateflow diagram (Gear_logic) a sebességváltást, a gázadás és a jármű sebessége alapján modellezi (lásd 131. Ábra). A *down_threshold* és az *up_threshold* kimenetek, a gázpedál és az aktuális sebességfokozat által kezelhető minimális, és maximális sebességértékeket jelentik. Elmondható, hogy ezen küszöbértékek átlépésével történik a sebesség fokozatának a váltása. A Simulink *calculate_thresholds* függvénye ezt a két értéket a gázadás és a sebességfokozat felhasználásával számítja ki, melyeket bemeneti paraméterként használ fel. Ha az aktuális sebesség, a TWAIT-nál hosszabb ideig nagyobb, mint az *up_threshold*, akkor a diagram magasabb fokozatba kapcsol. Ezzel szemben, ha az aktuális sebesség a TWAIT-nál hosszabb ideig alacsonyabb, mint a *down_threshold*, akkor a diagram alacsonyabb fokozatba vált. Javasolt a szimuláció futtatását időzítettre állítani, vagy lépésről-lépésre vizsgálni a modell működését, így megtudjuk figyelni az egyes sebességfokozatok közti átmenetet.



131. Ábra – Négyfokozatú sebességváltó logikai felépítése állapotblokkok segítségével (MathWorks, The MathWorks, Inc., 2021)

5.3.d. Példa – Elektromos áramköri átalakító

Következő példánkban, a már előzőkben említett felhasználási körök közül, az elektromos áramköri alkalmazásukat tekintjük hibrid rendszereknek egy átalakítón keresztül (Schaft & Schumacher, 2000) munkája alapján. A 132. Ábra egy áramkört ábrázol, ami egy L induktorból Φ_L mágneses fluxuskapcsolással, egy C kondenzátorból c elektromos töltéssel és egy R ellenállású terhelésből, valamint egy diódából és egy kapcsolóból áll, a következő kapcsolási állapotokkal ellátva:
$s = 1$ (kapcsoló zárva)
$s = 0$ (kapcsoló nyitva)



132. Ábra – Elektromos áramköri átalakító felépítése

A diódát ideális diódaként modellezzük, melynek konstitutív összefüggése röviden a következőképpen fejezhető ki:

$$v_D\, i_D = 0, \quad v_D \le 0, \quad i_D \ge 0 \tag{5.10}$$

Az áramkört arra használjuk, hogy az ellenállás terhelésén olyan feszültséget kapjunk, amely magasabb, mint a bemeneti E forrás feszültsége.

Egyértelmű, hogy a rendszer hibrid rendszerként ábrázolható négy helyzettel vagy ún. üzemmóddal, amelyek megfelelnek a dióda-karakterisztika két szegmensének és a két kapcsolóállásnak. Továbbá a nyitott kapcsolóval rendelkező helyről, a zárt kapcsolóval rendelkező helyre és fordítva, az átmenetek irányítottak, kívülről indukáltak, míg a dióda-karakterisztika egyik szegmenséről a másikra történő váltásnak megfelelő átmenetek autonómok. Folytonos állapotváltozóként (energia) a q_c elektromos töltést és a Φ_L mágneses fluxust, tárolt

energiaként pedig az $\frac{1}{2C}q_C^2 + \frac{1}{2L}\Phi_L^2$ kvadratikus függvényt tekintve az áramkör következő dinamikai egyenleteit kapjuk:

$$\begin{bmatrix} \dot{q}_C \\ \dot{\Phi}_L \end{bmatrix} = \begin{bmatrix} -\frac{1}{R} & 1-s \\ s-1 & 0 \end{bmatrix} \begin{bmatrix} \frac{q_C}{C} \\ \frac{\Phi_L}{L} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} E + \begin{bmatrix} si_D \\ (s-1)v_D \end{bmatrix}, \tag{5.11}$$

Ahol $s = 0, 1$ a kapcsolót jelöli, E a bemeneti forrás feszültsége, és i_D, v_D a diódán átfolyó áram, illetve az ideális diódán keresztüli feszültség. Az áramkör dinamikáját az utóbbi egyenlet teljesen meghatározza a kapcsoló helyzetével (diszkrét változó), és az ideális dióda által megadott konstitutív összefüggésével együtt.

A négy hely különálló dinamikáját a következő módon kapjuk meg:

1. hely: $s = 0, v_D = 0$
2. hely: $s = 1, i_D = 0$
3. hely: $s = 0, i_D = 0$
4. hely: $s = 1, v_D = 0$

Ez, a négy hely mindegyikére, a következő folytonos dinamikát eredményezi:

$$\begin{aligned} \dot{q}_C &= -\frac{1}{L}\Phi_L - \frac{1}{RC}q_C \\ \dot{\Phi}_L &= -\frac{1}{C}q_C + E \end{aligned}$$

$$\begin{aligned} \dot{q}_C &= -\frac{1}{RC}q_C \\ \dot{\Phi}_L &= E \end{aligned} \tag{5.12}$$

$$\begin{aligned} \dot{q}_C &= -\frac{1}{RC}q_C \\ \dot{\Phi}_L &= 0 \end{aligned}$$

$$\begin{aligned} \dot{q}_C &= 0 \\ \dot{\Phi}_L &= E \end{aligned}$$

Ahhoz, hogy megtaláljuk az aktuálisan aktív helyet, először megfigyeljük a kapcsoló helyzetét. $s = 0$ esetén az 1. és a 3. helyet kapjuk. Az 1. helyet a $v_D = 0$ és $i_D = \frac{\Phi_L}{L} \geq 0$ határozza meg. Az utóbbi egyenlőtlenségből adódik, hogy

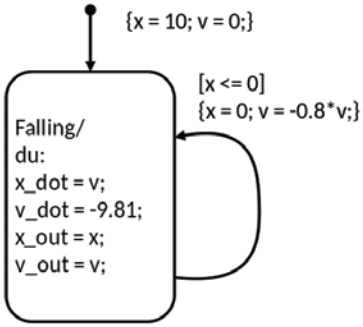
a $\Phi_L \geq 0$ hely invariáns. A harmadik hely az $i_D = \frac{\Phi L}{L} = 0$ és $v_D = E - \frac{q_C}{C} \leq 0$ által adott. Ebből adódik a $\Phi_L = 0$ és $q_C - E \geq 0$ hely invariáns. Hasonlóképpen, ha $s = 1$, akkor a rendszer a 2. helyen van $i_D = 0$ és a feszültség $v_D = -\frac{q_C}{C} \leq 0$, ami a $q_C \geq 0$ hely invariánst adja. S végül a 4. helyen van, ha $v_D = \frac{q_C}{C} = 0$ és $i_D \geq 0$ ami a $q_C = 0$ hely invariánshoz vezet.

Továbbá, egyszerű felírni az összes átmeneti feltételt és az ugrási relációkat. Valójában látható, hogy ha egy kezdeti folytonos állapotból indulunk ki, ahol $\Phi_L \geq 0$ és $q_C \geq 0$, $E \geq 0$ bemeneti feszültséggel, akkor nem történnek ugrások, és $\Phi_L(t) \geq 0$ és $q_C(t) \geq 0$ minden $t > 0$ esetén.

Fontos megjegyezni, hogy a 3. hely két helyváltozási invariánsának egyike, nevezetesen $q_C - E \geq 0$, explicit módon függ a külső folytonos E változótól. Így a példa beleillik az általánosított hibrid automata modellbe.

5.3.e. Példa - Pattogó labda modell

A pattogó labda modellje egy tipikus példája a hibrid dinamikus rendszernek. A klasszikus pattogó labda példája, egy előre meghatározott magasságból ledobott labda mozgásából indul ki. Egy bizonyos idő után a földre ér, energiát veszít, majd visszapattan a levegőbe, és ismét zuhanni kezd. Ez a fizikai jelenség a következő hibrid automatával ábrázolható:



133. Ábra – Hibrid automata a pattogó labda modell leírására (Ábrahám, 2019)

Diszkrét állapotban a feltételezett m = 1kg tömegű labda mozgását a következő differenciálegyenlet szabályozza:

$$\begin{aligned} \dot{x} &= v \\ \dot{v} &= -9.81 \end{aligned} \tag{5.13}$$

ahol x a labda talajtól mért magassága, v a labda sebessége, és $9.81 ms^{-2}$ jelöli a földi nehézségi gyorsulást. Az $x \geq 0$ biztosítja, hogy a labda mindig visszapattan, amikor földet ér. A pattogást modellező egyetlen diszkrét átmenetet $x = 0 \wedge v \leq 0$ feltétel biztosítja, amely feltétel „kimondja", hogy a pattogás, az esés után, a talajra érkezéskor következik be. A megfelelő $v := -c \cdot v$ visszaállítási feltétel, a labda deformációjából eredő energiaveszteséget veszi figyelembe, ahol $c \in [0,1]$ állandó (Ábrahám, 2019).

Jelen példa keretein belül két- és háromdimenziós ábrázolások segítségével mutatjuk be a HyEQ függvények működését, amely egy ilyen modell szimulációját – magába foglalva az áramlást, a leírt ívet és a fázistérben megvizsgált pályát – jeleníti meg. A BouncingBallSubsystem.m-ben definiálunk egy hibrid rendszert a HybridSystem osztály egy alosztályának létrehozásával, és a *flowMap, jumpMap, flowSetIndicator, jumpSetIndicator* függvények implementálásával. Részben ezen paraméterek definiálásával mutatja majd be a HyEQ toolbox-ot a 6.3.2-es alfejezetünk.

39. Forráskód – Pattogó labda modell

```
% BouncingBall - set C, D, f, g

classdef BouncingBallSubsystem < HybridSubsystem


  properties
    bounce_coef = 0.9;
    gravity = -9.8;
  end

  methods
    function obj = BouncingBallSubsystem()
% Constructor.
      state_dim = 2;
      input_dim = 1;
      output_dim = 2; % Matches default.
      output_fnc = @(x) x; % Matches default.
      obj = obj@HybridSubsystem(state_dim, input_dim,
```

```
18            output_dim, output_fnc);
19          end
20
21          function xdot = flowMap(this, x, u, t, j)
22            xdot = [x(2); this.gravity];
23          end
24
25          function xplus = jumpMap(this, x, u, t, j)
26            xplus = [x(1); -this.bounce_coef*x(2) + u];
27          end
28
29          function C = flowSetIndicator(this, x, u, t, j)
30            C = x(1) >= 0 || x(2) >= 0;
31          end
32
33          function D = jumpSetIndicator(this, x, u, t, j)
34            D = x(1) <= 0 && x(2) <= 0;
35          end
36        end
37      end
```

A modell számításának és grafikus kimeneteinek menetét az alábbiakban ismertetjük. Első lépésként a sys_bb változóba elmentjük a beépített BouncingBall példát tartalmazó attribútumokat.

sys_bb = hybrid.examples.BouncingBall();

A megoldás kiszámítása a slove függvény meghívásával történik, melynek paraméterei a system_bb-ben megadott kezdeti állapotok, és időtartam. Maga a függvény, eredményként, egy *HybridSolution* objektumot ad vissza, amely a megoldásra vonatkozó információkat tartalmazza:

```
HybridSolution with properties:

           x0: [2×1 double]
           xf: [2×1 double]
    termination_cause: J_REACHED_END_OF_JSPAN
            t: [5439×1 double]
```

```
            j: [5439×1 double]
            x: [5439×2 double]
    flow_lengths: [30×1 double]
     jump_times: [30×1 double]
shortest_flow_length: 0.0426
  total_flow_length: 8.2003
     jump_count: 30
```

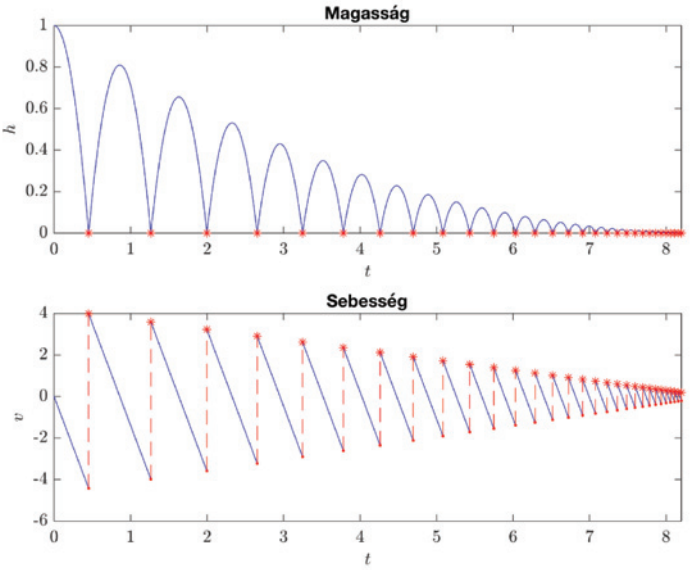
A kétdimenziós grafikus kimenet ábrázolására a *HybridPlotBuilder* osztály szolgál. Ezen belül van lehetőségünk az általános grafikus beállításokat is elvégezni, mint pl. a szín, a vonalstílusok, a jelölések és az ábrázoltatási sorrendek konfigurálása. Ezen utasítássorozatok tekinthetők meg a következő programkód *plot flows* részében. A leírt ívek ábrázolására a *slice* parancsot alkalmazzuk, amellyel egyszerre váltani tudjuk az ábrázoltatások sorrendjét slice([2, 1]), és amelynek eredménye egy 3D-s grafikus kimenet. A *slice* függvényt, a rendszer dimenziójának csökkentésére is használhatjuk, ha csak néhány komponenst adunk meg. Értelemszerűen a vizuális eredmény ez esetben is a t és j paraméterek függvényében értelmezhető.

A modell ábrázolását leíró programkód (MathWorks, The MathWorks, Inc., 2021), valamint az ebből kapott grafikus kimenetek:

40. Forráskód – Pattogó labda modell - HyEQ

```
1   % Hybrid Bouncing Ball example - HyEQ
2   x0 = [1;0];
3   tspan = [0, 20];
4   jspan = [0, 30];
5   config = HybridSolverConfig('Refine', 32); % 'Refine'
    makes plots                    % smoother.
6
7   sol_bb = sys_bb.solve(x0, tspan, jspan, config)
8
9   % Plot flows
10  figure()
11  plot_builder_bb = HybridPlotBuilder();
12  plot_builder_bb.subplots('on')....
13    .titles("Height", "Velocity") ...
14    .labels('$h$', '$v$') ...
15    .plotFlows(sol_bb);
16
```

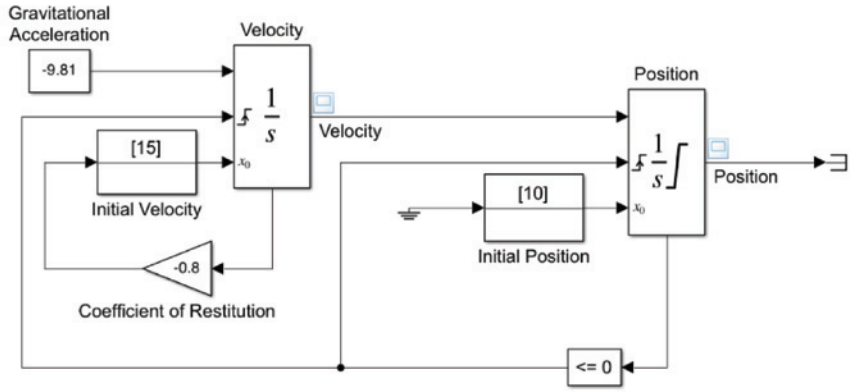
```
17      % Plot Hybrid Arcs
18      figure(2)
19      plot_builder_bb.subplots('on')...
20        .titles("Height", "Velocity") ... % The order
        of the titles
21        %  must match the order of the components in the
        solution.
22        .slice([2, 1]) ... % Switch the order
23        .plotHybrid(sol_bb);
24
25
26
27      % Plot State
28      figure(3)
29      plot_builder_bb = HybridPlotBuilder();
30      plot_builder_bb.title("Phase Space: $v$ vs. $h$")
        ...
31        .labels('$h$', '$v$') ...
32        .plotPhase(sol_bb)
```



134. Ábra – Pattogó labda modell szimulációja

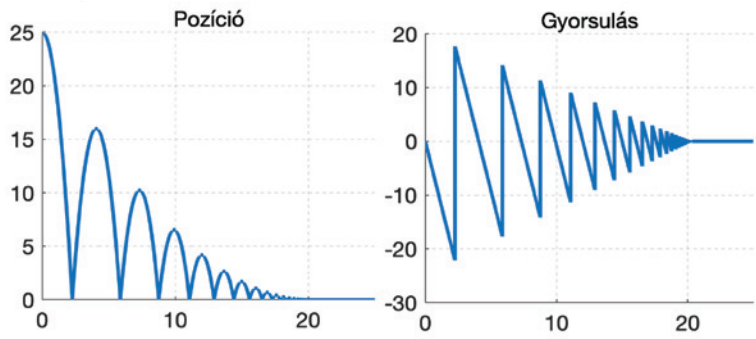
A pattogó labda az egyik legegyszerűbb modell, amely a Zénó-jelenséget mutatja. A Zénó-viselkedést úgy jellemezhetnénk, hogy bizonyos hibrid rendszerek esetében véges időintervallumban végtelen számú esemény következik be. Ahogy a labda energiát veszít a pattogó labda modellben, az egymás utáni talajjal való ütközések nagy száma egyre kisebb időintervallumokban kezd bekövetkezni. Ezért a modell, Zénó-viselkedést mutat. A Zénó-viselkedéssel rendelkező modelleket, nehéz számítógépen szimulálni, de számos gyakori és fontos mérnöki alkalmazásban előfordulnak.

Most tekintsük meg a pattogó labda modelljének egy lehetséges felépítését, és annak grafikus kimenetét a Simulink rendszeren belül.



135. Ábra – Pattogó labda modell megvalósítása Simulink-ben

Egy pattogó labda modellezéséhez két *Integrátor* blokkot használhatunk. A bal oldali *Integrátor* a sebesség meghatározásáért felelős, a jobb oldali pedig a pozíciót írja le. Fontos meghatározni, hogy a pozíciót leíró *Integrátor*-blokk alsó határa nulla legyen. Ez a feltétel, azt a korlátot jelenti, hogy a labda nem mehet a talaj alá. A pozícióintegrátor állapotportját, és a megfelelő összehasonlítás eredményét arra használjuk, hogy érzékeljük, amikor a labda a földre ér, és mindkét integrátort visszaállítsuk.



136. Ábra – Pattogó labda modell grafikus ábrázolása

Bővebben a pattogó labda modellről és annak variánsairól (pl. pattogó labda mozgó platformon) Sanfelice *HyEQ:* A *Toolbox for Simulation of Hybrid Dynamical Systems* című munkájában olvashatunk (Sanfelice, Nanez, & Copp, 2013).

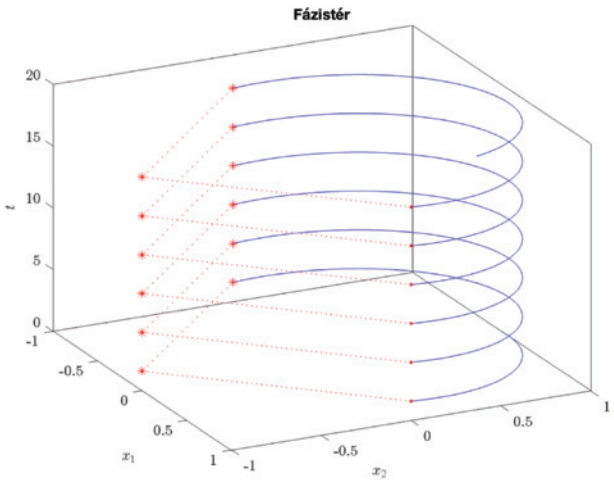
5.3.f. Példa - Állapotábrázolás 3D-s térben

A következőkben terjesszük ki az 5.3.e. példában ábrázolt hibrid rendszert, háromdimenziós térbe. Erre a beépített *Example3DHybridSystem()* modellt alkalmaztuk. A modell így, egy újabb paraméterrel bővült - jelölje *config* -, amely meghívja a *HybridSolverConfig()* függvényt, majd bemeneti paraméterként szolgál a számítást végző solve metódusnak. A *hybrid.plotFlowLengths* függvény kirajzolja az egyes áramlási szakaszok (intervallumok) hosszát, egy adott megoldási objektumhoz. Ez a függvény nem támogat semmiféle testreszabást, csupán egy bemeneti paraméterrel kell ellátni, mely bemeneti paraméter esetünkben, a kiszámított 3D-s értékek.

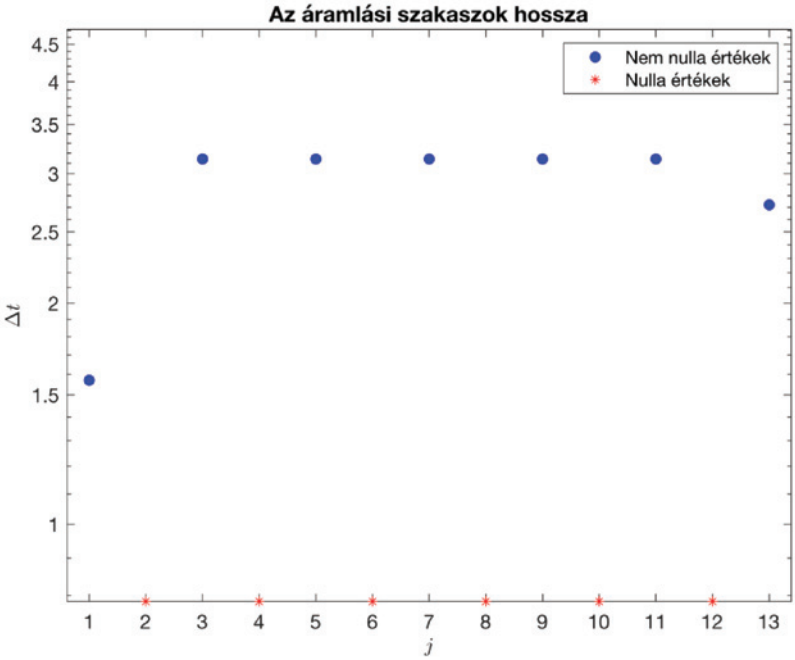
41. Forráskód – Állapotábrázolás 3D-s térben

```
1       % Plot State for 3D System
2       figure(4)
3       clf
4       system_3D = hybrid.examples.
        Example3DHybridSystem();
5       x0 = [0; 1; 0];
6       tspan = [0, 20];
```

```
7       jspan = [0, 100];
8       config = HybridSolverConfig('MaxStep', 0.1);
9       sol_3D = system_3D.solve(x0, tspan, jspan,
        config);
10
11      HybridPlotBuilder().title("Phase Space") ...
12        .labels("$x_1$", "$x_2$", "$t$") ...
13        .jumpLineStyle(":") ...
14        .plotPhase(sol_3D)
15      view([63.6 28.2])
16
17      figure(5)
18      hybrid.plotFlowLengths(sol_3D)
```



137. Ábra – HyEQ modell 3D-s kimenete



138. Ábra – Az áramlási szakaszok hossza

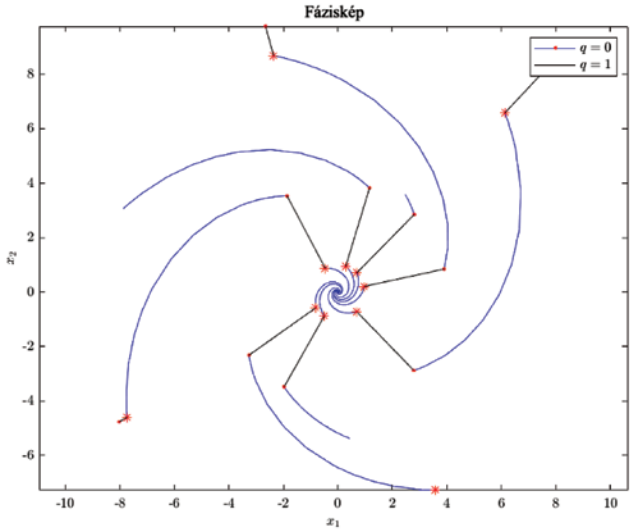
5.3.g. Példa - Rendszermodellek ábrázoltatása

Ahogyan az alapértelmezett *plot* ábrázoltatási funkción belül, úgy a hibrid rendszerek esetében is konfigurálható a grafika egyes részeinek a megjelenítése. Ez a példa arra mutat rá, hogy milyen módon változtatható a hibrid grafika egyes részeinek megjelenítése a plot-on belül. A *system_with_modes* rendszer, egy folytonos értékű $z \in \mathbb{R}^2$ változóból, és egy diszkrét értékű $q \in \{0, 1\}$ változóból áll.

A már előző példánkban is alkalmazott *slice*([1, 2]) parancs segítségével, kihagyjuk a q változót az ábrázolásból. Ezt követően létrehozunk egy q tömböt, amely minden egyes időbeli lépésnél tartalmazza a q értékét. Végül a *filter* $(q == 0)$ meghívása lehetővé teszi, hogy csak azokat az időlépéseket ábrázoljuk, ahol $q=0$.

```
% Plotting System Models
figure(6);
% A 3D system with a continuous variable z \in \
reals^2
% and a discrete variable q \in {0, 1}
system_with_modes = hybrid.examples.
ExampleModesHybridSystem();

for i=1:7
  % Create a random initial condition and solve.
  z0 = 20*rand(2, 1) - 10;
  q0 = round(rand());
  sol_modes = system_with_modes.solve([z0; q0], [0,
10], [0, 10],
  " silent");

q = sol_modes.x(:, 3);

  % Plot the [1, 2] components (that is, the first two
  % components) of sol_modes at all time steps
  % where q == 0.
  builder = HybridPlotBuilder();
  builder.title( "Phase Portrait") ...
    .labels( "$x_1$",  "$x_2$") ...
    .legend( "$q = 0$")...
    .slice([1,2]) ... % Pick which state components to
            % plot
    .filter(q == 0) ... % Only plot points where q
is 0.
    .plotPhase(sol_modes)
  hold on
  % Plot in black the solution (still only the
[1,2]
  components) for all time steps where q == 1.

  builder.flowColor( "black") ...
```

```
32          .jumpColor( "none") ...
33          .legend( "$q = 1$")...
34          .filter(q == 1) ... % Only plot points where q
        is 1.
35          .plotPhase(sol_modes)
36      end
37      axis equal
```



139. Ábra – Rendszermodell ábrázolása q=0 feltétel esetén

A pattogó labda modell példában egy alosztály keretein belül bemutattuk az áramlási térkép, az áramlási halmaz, az ugrási térkép és az ugrási halmaz implementálásának lehetőségeit, azaz *C, D, f* és *g* megadási módját. Egy új alosztályának a létrehozása helyett, van egy gyorsabb módja is a hibrid rendszerek létrehozásának. A *HybridSystemBuilder* keretein belül meghatározhatjuk az előzőkben felsorolt függvények értékeit. A MATLAB-ban található HyEQ Toolbox használatát, valamint a *C, D, f* és *g* paraméterek leírását a 6.3.2-es fejezet ismerteti.

43. Forráskód – HybridSystemBuilder alkalmazása

```
% Quick System Definition
figure(7)
system_inline = HybridSystemBuilder() ...
      .f(@(x, t) t*x) ... % the function arguments can
                  % by (x), (x,t), or (x, t, j).
      .g(@(x) -x/2) ...
      .C(@(x) 1) ...
      .D(@(x) abs(x) >= 1) ...
      .build();
sol_inline = system_inline.solve(0.5, [0, 10], [0, 10]);
plotFlows(sol_inline)
```

Ez a módszer lassabban megoldható és nehezebben hibakereshető rendszereket hoz létre, ezért inkább egyszerű rendszerek teszteléséhez ajánlott.

6. MODELLEZÉS ÉS SZIMULÁCIÓS ESZKÖZÖK

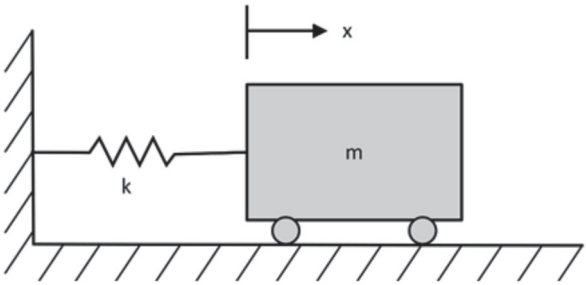
Szimulációk kivitelezésére manapság számos eszköz áll rendelkezésre. Az egyik legismertebb a MATLAB. A program, illetve az azt működtető programozási nyelv, számos lehetőséget hordoznak magukban. Annak érdekében, hogy a modellalkotás könnyebb legyen, specializált eszköztárakat telepíthetünk fel. A következőkben ezek közül néhány, grafikus felülettel is ellátott, toolboxot fogunk bemutatni (Inc., MathWorks, 2021).

6.1. Simulink

A Simulink a MATLAB egy olyan bővítménye, amely lehetővé teszi a felhasználók számára, hogy gyorsan és pontosan felépítsék a dinamikus rendszerek számítógépes modelljeit blokkdiagramok segítségével. A Simulink-el könnyen modellezhetők összetett nemlineáris rendszerek. A Simulink-modell tartalmazhat folytonos és diszkrét idejű komponenseket, emellett képes grafikus animációkat készíteni, amelyek vizuálisan mutatják be a szimuláció előrehaladását, jelentősen javítva a rendszer viselkedésének megértését.

A múltban egy dinamikus rendszer számítógépes modelljének kifejlesztéséhez, gyakran használták azt a megközelítést, hogy egy blokkdiagramal kezdték a modellezést. Ezután a blokkdiagramot lefordították egy programozási nyelv forráskódjába. Ez a gyakorlat egyrészt kétszeres erőfeszítéssel járt, mivel a rendszert és a vezérlőt kétszer kellett leírni – egyszer blokkdiagram formájában, majd újra a programozási nyelven – másrészt azt a kockázatot is magán hordozza, hogy a blokkdiagramról a forráskódra történő fordítás pontatlan lehet. A vezérlőrendszer-tervezés hibakeresése során problémát jelentett, a hiba helyének meghatározása. A hiba lehetett a tervezésben (blokkdiagram világa), a programban (programozási nyelv világa), vagy a blokkdiagramról a programra történő fordításban. A Simulinkkel megszűnik a modell programozási nyelven történő újraírásával járó megkettőzött munka, és az a tény, hogy a „program” maga a blokkdiagram, kiküszöböli annak kockázatát, hogy a program esetleg nem pontosan valósítja meg a blokkdiagramot.

A programozásnak ez a blokkdiagramos megközelítése, drámai termelékenységnövekedést eredményezhet. Példaként tekintsük meg az alábbi egyszerű rugós-tömeges rendszert.



140. Ábra – Egyszerű rugós-tömeges rendszer

A 2. Táblázat összehasonlítja az ábrázolt rendszert (140. Ábra) 8086-os Assembly nyelven 16 bites egészértékű aritmetikával és egyszerű Euler-integrációval; FORTRAN-ban lebegőpontos aritmetikával; MATLAB-ban mátrix-aritmetikával; és Simulinkben modellező programokat. A Simulink program esetében, a program utasításai helyett, a blokkok számát soroljuk fel, és az egérkattintások számát az billentyűleütések számával együtt szerepeltetjük.

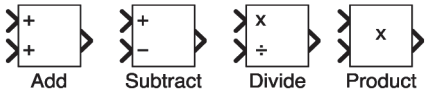
3. Táblázat – Simulink sebességének összehasonlítása

Programozási nyelv	Kódsorok/blokkok száma	Hozzávetőleges billentyűleütések
8086 Assembly	92	1540
FORTRAN	14	240
MATLAB	3	90
Simulink	4	25

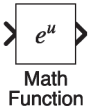
6.1.1. Alapvető Simulink blokkok bemutatása

A *Constant* egy konstans érték tárolására alkalmas blokk. Amennyiben egy a *workspace*-en létező változó nevét adjuk meg értékül, képes átvenni azt.

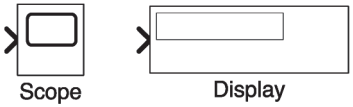
Alapvető matematikai műveletek (összeadás, kivonás, osztás, szorzás) elvégzésére, az alábbi blokkok használhatóak. A bemenetek száma minden esetben módosítható tetszőleges mennyiségűre, illetve sorrendűre.



Matematikai műveletek elvégzésére a *Math Function* blokk alkalmas.



Ebben a blokkban több művelet közül lehet választani, mint például hatványozás, logaritmizálás, gyökvonás, stb.
Az értékek megjelenítésére leggyakrabban használt blokkok a *Scope* és a *Display*. Utóbbi a szimuláció lefutása utáni értékeket, míg az előbbi a futás közbeni eredményeket is képes kijelezni az idő függvényében.



Az *Integrator* blokk a bemeneti jel időbeli integráljának értékét számolja ki. Míg ezek az egyenletek folyamatos időben pontos összefüggést határoznak meg, a Simulink numerikus közelítő módszereket használ a véges pontosságú kiértékelésükhöz. A felhasználó a Simulink-ben több, különböző numerikus integrációs módszert is használhat a blokk kimenetének kiszámításához. A *Konfigurációs paraméterek* párbeszédpanel eléréséhez, duplán kell a blokkra kattintani.

Az *Unit Delay* blokk a bemenetét, a megadott mintaidőszakkal tartja és késlelteti. Egy diszkrét időben történő szimulációban, ahol a rendszer viselkedése időlépésről időlépésre változik, a *Unit Delay* blokk segít rögzíteni és továbbítani az előző időpillanatbéli értéket. Ez a blokk egyenértékű a z^{-1} diszkrét idejű operátorral. A blokk egy bemenetet fogad el, és egy kimenetet generál. Minden jel lehet skalár vagy vektor.

A blokk kimenetét az első mintavételi periódusra a *Kezdeti feltételek* paraméterrel adja meg. Ennek a paraméternek a gondos megválasztásával minimalizál-

ható a nem kívánt kimeneti viselkedés. A mintavételek közötti időt a *Sample time* (mintavételi idő) paraméterrel adja meg. A -1-es beállítás azt jelenti, hogy a blokk örökli a *Sample time* (mintavételi idő) értéket.

6.2. SimEvents

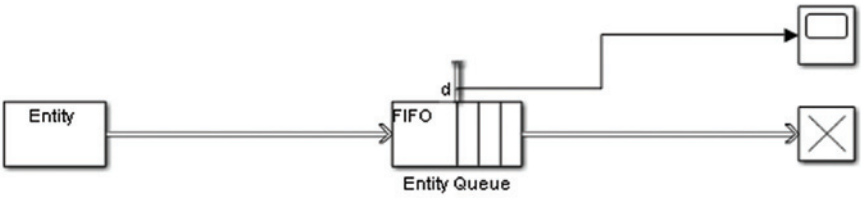
A SimEvents egy diszkrét esemény-szimuláló motor, és komponens könyvtár, ami esemény vezérelt rendszerek analizálására és optimalizálására lett kifejlesztve. Ennél a megközelítésnél maga az esemény a fő szempont, ami – folytonos és időbeli modellektől eltérően – különböző helyzetek és paraméterek megfigyelésére teszi alkalmassá. Az előre elkészített blokkok lehetőséget adnak egyszerű és komplex modellek elkészítésére (például útválasztás, késés kezelés és prioritások ütemezése témában). A SimEvents betekintést ad az időzítés és erőforráskezelés teljesítményre gyakorolt hatásaiba, ami egy kiemelten fontos szempont hardver-, szoftverarchitektúráknál, kommunikációs hálózatoknál, tömegkiszolgálásnál és ipari ellátási lánc tervezésénél.

6.2.1. Alapvető SimEvents blokkok bemutatása

Entity Generator – Nevéből is adódóan, entitásokat hoz létre. Ezek diszkrét egységek, amelyek a diszkrét eseményszimulálás egyik elengedhetetlen részét alkotják. Az entitások hordozhatnak skaláris, struktúra(busz), vagy vektor adatokat és az értelmezésük modellenként eltérő. Entitások lehetnek ügyfelek sorakozás közben, adatcsomagok, vagy bármi, ami a modell szimulálásához szükséges. Alapértelmezetten az entitások generálása időalapú, és a periódus megadásával lehet meghatározni a generálásukat. Lehetséges esemény alapú entitások generálása is, ebben az esetben egy külső esemény váltja ki azok létrehozását.
Queue, Entity Queue – A blokk entitásokat vagy üzeneteket tárol érkezés/prioritás alapján. A blokk következő eleme elhagyja a sort, amint a folyamatban utána levő elem képes fogadni azt. A két objektum között a különbség abban nyilvánul meg, hogy milyen alapértelmezett értéket használnak a legöregebb elem felülírására a sor telítettsége esetén. A sor tulajdonságai között meghatározható a sor kapacitása, illetve az eljárás a megtelt sornál. A blokk 3 különböző sorbanállási módszert ismer: FIFO (first-in-first-out), LIFO (last-in-

first-out) és prioritás. További fontos tulajdonsága a Queue blokknak a kapacitás, ami azt határozza meg, hogy hány entitást képes fogadni, azaz milyen hosszú lehet a sor. Alapértelmezetten végtelen hosszúságú.
Entity Terminator – Ez a blokk fogadja és megsemmisíti az entitásokat. Arra szolgál, hogy szimbolizálja a modellt elhagyó entitásokat, például vásárló elhagyja az üzletet a vásárlás befejeztével.
Scope – Bár ez általános Simulink építőelem, ennek ellenére említése fontos a SimEvents tárgyalása közben, mivel ez a blokk képes az épített rendszer információinak megjelenítésére. A Scope blokk más objektumok kimeneti ágára kapcsolható, a szükséges adatok megjelenítéséhez.

Az előbb felsorolt 4 blokk elegendő egyszerűbb SimEvents modellek építéséhez, ami később további modellek alapjául szolgál és bővíthető. Az alapmodell periodikusan entitásokat generál, a létrehozott egyedek belépnek a sorba, ahonnan tovább haladva megszüntetésre kerülnek. Annak érdekében, hogy a felhasználó betekintést kapjon, mi történik az összeállított rendszerben, egy Scope blokk kapcsolódik a rendszerre. Érdemes megfigyelni, hogy az entity generátor, sor és kilépési pont összekötését dupla vonallal, míg a Scope blokk bekötését szimpla vonallal jelöli, ezzel segítve a funkcionalitás szerinti megkülönböztetést és áttekinthetőséget (lásd 141. Ábra) (MathWorks, The MathWorks Inc., 2022).



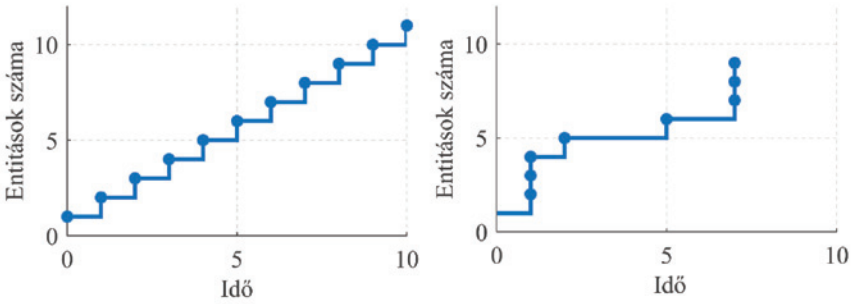
141. Ábra – Egyszerű sorbanállási modell, sort elhagyó egyedek számolásával

Mivel a valóságban, legtöbb esetben az entitások (vásárlók, ügyfelek, erőforrások stb.) nem azonos időközönként, periodikusan érkeznek, hanem valamiféle véletlen eloszlást mutatnak, az alapmodell gyorsan módosítható a generátor paramétereinek változtatásával. Az egyszerűség kedvéért, az entitások létrehozását tartalmazó ablakban az idő forrása legyen ***MATLAB action***, és a rendelkezésre álló szerkesztőben lehetséges különböző paramétereket beállítani függvények segítségével.

44. Forráskód – Véletlen számok generálása

```
1                    dt = rand(1,1);
```

A fenti parancs 0 és 1 között generál $1 \cdot 1$ méretű tömbbe, egyenletes eloszlású véletlen számot. A 142. Ábrán megfigyelhető az egyenletes eloszlás rendezettsége, 1 időegységenként (x tengely) 1 entitás lép be, és hagyja el a sort. Ezzel szemben véletlen eloszlást használva az entitások között eltérő időegységek figyelhetők meg.



142. Ábra – Periodikus és egyenletes eloszlású véletlen érkezési idő közti különbség

A ***Statistics and Machine Learning Toolbox*** telepítésével elérhetővé válnak az *exprnd()* és *poissrnd()* függvények, amelyek gyakran használt függvényekkel bővítik a készlettárat.

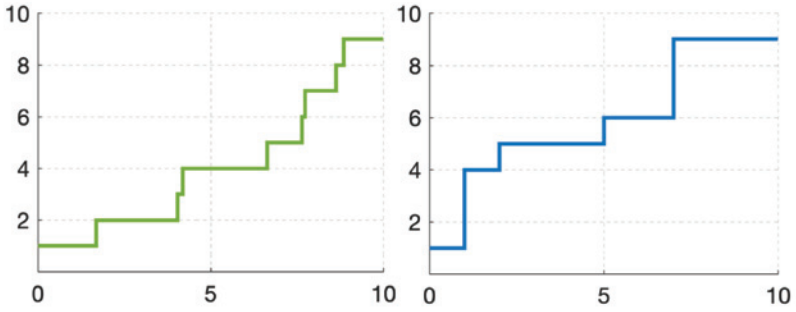
45. Forráskód – Exponenciális eloszlású véletlen számok generálása

```
1
2
        mean = 1;
3
4       dt = -mean*log(1-rand());
5
6       %alternatív parancs az említett bővítmény használata
        esetén
7       dt = exprnd(1);
```

46. Forráskód – Poisson eloszlású véletlen számok generálása

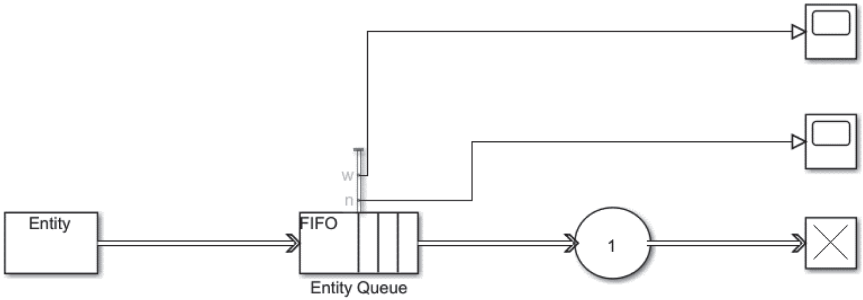
```
persistent rngInit;
if isempty(rngInit)
  seed = 12345;
  rng(seed);
  rngInit = true;
end

% Minta: Poisson eloszlás
% m: Mean
m = 1;
dt = poissrnd(m);
```

Entity Server – Az eddig elkészített rendszerben létrejönnek az egyedek, beállnak a sorba, majd megszűnnek. A kiszolgáló blokk beiktatásával kiegészítve azonban, már egy primitív tömegkiszolgáló rendszer jön létre. DEVS szimuláció során, ez a blokk tárolja bizonyos időtartamig az entitásokat, amit szolgálati időnek hívnak. A kiszolgáló blokk képes több entitás kiszolgálására azonos időben, és a szolgálati idő leteltével megpróbálja az entitás(oka)t kiengedni a kimeneten. Az objektum lehetőséget ad entitások közti előzés szimulálására, ebben az esetben a kiszolgált egyed egy második kimeneten át távozhat.



143. Ábra – Exponenciális és Poisson eloszlású véletlen érkezési idő közti különbség

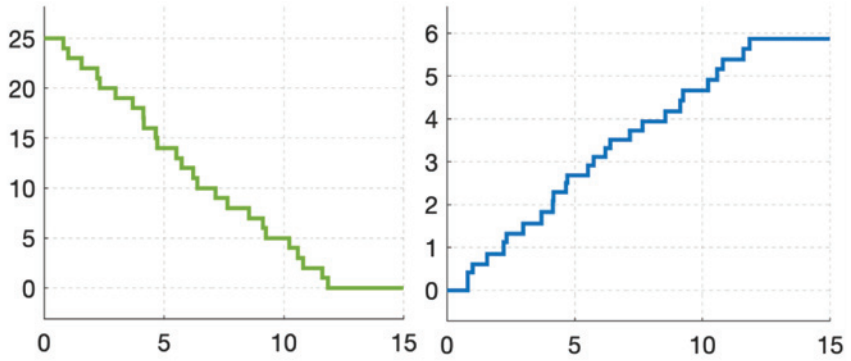
Az alapértelmezett kiszolgáló 1 entitást képes ellátni és 1 időegységet vesz igénybe a szolgáltatás, ebből következően, a sor elhagyását vizsgálva, ismét erősen szabályozott eredményt ad vissza. Lehetőség van robbanásszerű entitásgenerálásra is. Ez a módszer azt jelenti, hogy a szimuláció indításakor több entitást generál le az erre szolgáló blokk, ezzel azonnal hosszabb várakozó sort kialakítva. Jó, életből származó példa erre, a már nyitás előtt sorakozó vásárlók és ügyfelek.



144. Ábra – Kiindulási sor generálása egy kiszolgálóegység esetén

47. Forráskód – Robbanásszerű kezdeti entitásgenerálás (N=25)

```
1       N = 25;
2
3       persistent dtArray index
4       if isempty (dtArray)
5         dtArray = [zeros(1,N) inf];
6         index = 1;
7       end
8       dt= dtArray(index);
9       index = index + 1;
```



145. Ábra – Sorban álló entitások száma (zöld) és átlagos várakozási idő (kék)

6.2.2. További SimEvents blokkok bemutatása

Entity Input/Output Switch – 1-től 127-ig terjedő mennyiségű bemenettel rendelkező blokk. Elsődleges feladata, több különböző forrásból érkező entitások egy csatornába (kimenetre) terelése. Kapcsoló révén, tulajdonságai között beállítható, hogy az összes bemenetről fogadjon entitásokat, vagy valamilyen váltási szabály alapján működjön a folyamat. Az Output Switch ezzel szemben kimenti blokk, amely hasonló szerepkört lát el, viszont csak 1 bementtel és több kimenettel rendelkezik.
Entity Store – Entitások rendezetlen tárolására alkalmas, ahonnan engedély esetén azonnal az összes entitás továbbléphet. Ha a kimenet zárva van, a függőben lévő entitás várakozik a kimenet felszabadulásáig.
Repeating Sequence – Idő és kimeneti értékek megadásával periodikusan ismétlődő számsor kifejezése lehetséges általa.
Entity Replicator – Blokkból való kilépés előtt másolatot hoz létre az áthaladó entitásokról. Beállítható a létrehozandó másolatok száma, illetve hogy azok melyik kimeneten haladjanak tovább. Az eredeti egyed visszatartható, amíg a másolatok elhagyják a blokkot.
Resource Pool – Tetszőleges, modellhez szükséges erőforrások létrehozását teszi lehetővé. A beállítások között kiemelten fontos a név paraméter, hiszen a későbbiekben, ezen néven lehetséges elérni a blokkot. További beállításai között megadható a mennyisége, hogy egész egységekben vagy tört mennyiségben kerül használatra, illetve, hogy használat után visszakerül-e a rendszerbe.
Diszkrét egység – jármű, szerszám, alkatrész stb.
Tört mennyiség – nyersanyagok, áram, víz stb.

Resource Acquirer – Ez a blokk jelzi, hogy adott entitás igény tart az erőforrás használatára és megfelelő körülmények között, a használatra engedélyt is kap. Az erőforrás használatáért kialakulhat várakozási sor.
Resource Releaser – Az erőforrás paramétereitől függően vagy visszakerül a készletbe, vagy eltávozik a rendszerből.
Display – Kijelző, főként számadatok és információk megjelenítésére használatos, formátuma állítható.
Simulink Function – A blokk saját függvény írását teszi lehetővé. Felhasználása ezáltal sokrétű. A függvény megadása után, a szerkesztő, automatikusan létrehozza a szükséges változókat és blokkokat. Kiemelten fontos tudatosítani, hogy a függvény határozza meg a bemenetek és kimenetek számát. Kezelése hasonló az alrendszerekéhez: az objektumon való duplakattintás után tovább bővíthetők, illetve rendezhetők a függvény elmei.
Function Caller – Működése szorosan kapcsolódik az előbbi blokkhoz, hiszen abból ki lehet húzni, vagy üres objektumként deklarálni és megadni a függvény nevét. Funkcionalitásában, portok számában megegyezik a megadott függvénnyel és beépíthető a rendszerbe.

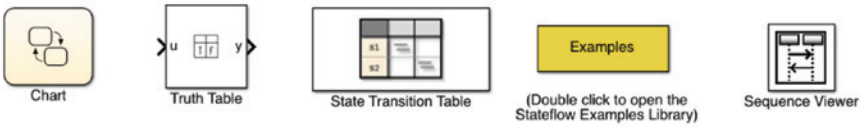
6.3. Stateflow

A MATLAB-on belül elérhető Stateflow, olyan grafikus nyelvet biztosít, amely állapotátmeneti diagramokat, táblázatokat, folyamatábrákat és igazságtáblázatokat tartalmaz. A Stateflow segítségével leírhatjuk, hogy miként reagálnak a MATLAB algoritmusok és a Simulink modellek a bemeneti jelekre. A Stateflow lehetővé teszi felügyeleti vezérlés, feladatütemezés, hibakezelés, kommunikációs protokollok, felhasználói felületek és hibrid rendszerek tervezését és fejlesztését. Segítségével kombinatív és szekvenciális döntési logikát modellezhetünk. Ezek Simulink modellen belül, blokkok segítségével szimulálhatók, de van lehetőségünk a MATLAB-on belül objektumként kezelni őket. A grafikus animáció lehetővé teszi a logika elemzését és hibakeresését végrehajtás közben.

A Stateflow-ot és az azon belül található blokkokat több módon is elérhetjük. Elsősorban meg kell nyitnunk egy üres Simulink modellt, majd a Library Browser-en belül a baloldalsó menüsorban ki kell választuk a Stateflow könyvtárat. A másik lehetőség, ha a MATLAB-ban lévő Command Windowba közvetlen utasításként megadjuk a *stateflow* parancsot. Ez azt eredményezi, hogy a program megnyitja a Simulinket és közvetlen a Stateflow könyvtáron belüli blokkokat (sflib) kínálja fel a felhasználó számára.

6.3.1. Alapvető Stateflow blokkok bemutatása

A következőkben ismertetjük a sflib-ben található blokkokat és azok funkcióit. A 146. Ábra szemlélteti a sflib könyvtárból elérhető blokktípusok ikonjait.



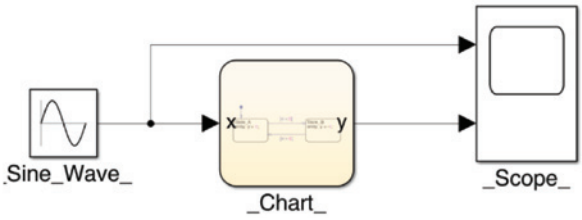
146. Ábra – Az sflib-ben található blokkok.

Állapot blokk – Chart

Segítségével véges állapotú eseményvezérelt rendszereket ábrázolhatunk. Eseményvezérelt rendszerek alatt az értendő, hogy egy bizonyos esemény hatására a rendszer átlép egyik állapotból a másikba. Ez az átmenet akkor következik be, ha a változást meghatározó feltétel igaz. Elmondható, hogy az állapotok és az átmenetek alkotják a rendszer alapelemeit. Lehetőségünk van állapotmentes folyamatábrák ábrázolására is. A Chartot duplakattintással nyithatjuk meg, és azon belül további blokkokat köthetünk össze, amelyek valamilyen kimenetet eredményeznek.

Bár a Chart blokk alapértelmezetten nem rendelkezik portokkal (147. Ábra), lehetőségünk van bemeneti (input) adatok rácsatolására és kimeneti (output) adatok lekérésére. Amennyiben bemeneti adatokat kapcsolunk a Chart blokkunkhoz, az automatikusan beviteli portokat hoz létre. *N* darab bemeneti adat csatlakoztatása esetén *N* darab port jön létre. A Charton belüli blokkok kimenete pedig a Chart kimeneti portjait és azok darabszámát befolyásolja.

Tekintsünk meg egy egyszerű modellt a Chart blokk alkalmazására. A modellünk tartalmaz Simulink és sflib blokkokat egyaránt (147. Ábra).



147. Ábra – Szinuszos hullám szimulálására alkalmas modell felépítése

Az egyes blokkok funkcióit az alábbiakban ismertetjük:

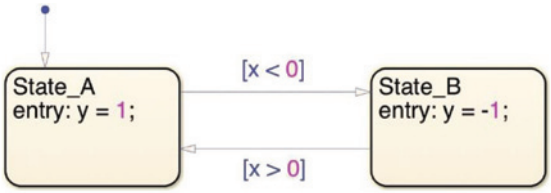
- **Sine Wave:** A bemeneti jelet biztosítja a Chart számára. Közvetlen, analóg jeleket küld a Scope-nak. Képes felhasználni a szimulációs időt. Elérhetőség – Simulink.
- **Chart:** Állapotokat és a köztük lévő átmeneteket biztosítja (1, -1). A Sine Wave által kapott jelet feldolgozva, az állapotok szerint digitális jelet küld a Scope számára. Elérhetőség – Stateflow.
- **Scope:** Feladata a grafikus megjelenítés. Elérhetőség – Simulink.

Most pedig vizsgáljuk meg a Chart-on belüli állapotok kapcsolatát (148. Ábra). Állapotokat lehelyezni, a Chart szerkesztőjében található baloldalsó menüsorból, a *State* lehetőséget kiválasztva tudunk. Jelen esetben két állapotról beszélhetünk: -1 és 1. Ennek megfelelően két állapotot hozunk létre *State_A* és *State_B* elnevezéssel. Az egyes állapotokat felruházhatjuk értékekkel. Példánk során az A állapot 1, a B állapot pedig -1 értéket szeretnénk, hogy felvegyen. Az értékadáshoz alkalmaznunk kell a következő sablont:

entry: változó_neve = érték;

Ezt követően meg kell határoznunk az állapotok közötti átmenetet. Ehhez az egyes állapotok blokkjainak széléből egyszerűen, kurzor segítségével nyilakat húzunk, hasonlóképp, mintha Simulink blokkokat kötnénk össze egymással. Az átmeneti kapcsolatok megadásakor van lehetőségünk meghatározni az átmeneti feltételeket.

A baloldalsó menüben található *default transition* objektum (kék pontból kiinduló nyíl) segítségével állíthatjuk be a kezdő állapotot. Ezen példánk esetében ez a *State_A*.



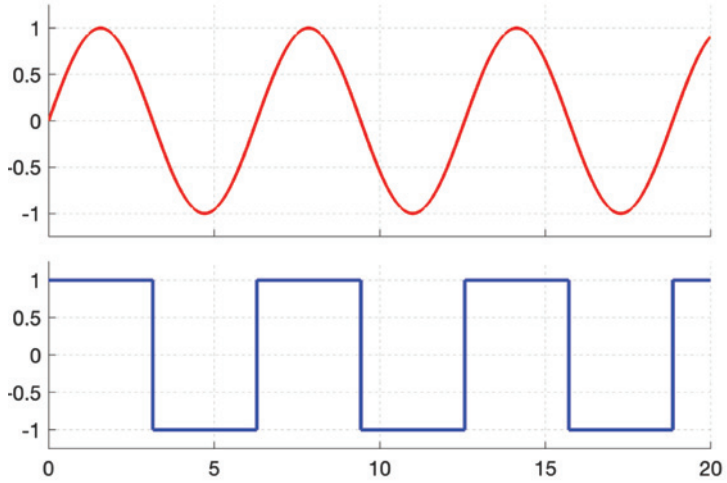
148. Ábra – Két állapot és a köztük lévő átmeneti feltétel.

Ahhoz, hogy a Chart képes legyen ki- és bemeneti értékekként kezelni az *x* és *y* paramétereket, be kell állítanunk azok adattípusát és fajtáját. Ezt a *Symbols*

Pane menüponton belül tudjuk megtenni. A Stateflow Chartjában található adattípusok a következők lehetnek:

- bemeneti *(input data)*
- kimeneti *(output data)*
- konstans *(constant data)*
- lokális *(local data)*
- paraméteres *(parameter data)*
- adattároló *(data store memory)*
- ideiglenes *(temporary data)*

Ezen paraméterek beállítását követően hozza létre a program a kimeneti és bemeneti portokat a Chart blokkon. A kimeneti portot összekötve a Scope bemenetével, elindíthatjuk a szimulációt a felső menüben lévő *Run* gombra kattintva. Stateflow modellek esetén a szimuláció idejét lassúra állítva, vagy lépésről lépésre érdemes megtekinteni, így könnyű áttekintést nyújt, hogy épp melyik állapot, vagy átmenet van érvényben. A vizsgálni kívánt időintervallumot szintén a felső menüben található *Stop time* segítségével állíthatjuk be. A végeredményt a Scopera való duplakattintással tekinthetjük meg. Jelen modellünk grafikus kimenetét mutatja be a 149. Ábra. A felső réteg a Sin Wave blokk grafikus kimenetét, az alsó réteg, pedig a Stateflow könyvtárhoz tartozó Chart állapotainak kimeneteit ábrázolja.



149. Ábra – Szinuszos hullámok grafikus eredménye. Felül Sine Wave (Simulink) piros színnel, alul Chart (Stateflow) állapotainak kimenete, kék színnel jelölve.

Igazságtábla - Truth Table

A Truth Table blokk egy igazságtábla-függvény, amely a MATLAB-ot használja műveletnyelvként. Az igazságtáblázatok a kombinatorikus logikai tervezést tömör, táblázatos formában valósítják meg. Amennyiben igazságtáblázat logikáját szeretnénk használni egy Simulink modellen belül, szükséges a Stateflow alkalmazása. A Truth Table blokkra való duplakattintással, megnyílik az igazságtábla szerkesztője (150. Ábra).

Condition Table

	DESCRIPTION	CONDITION	D1	D2	D3
1	Example condition 1	u >= 0	T	F	-
2	Example condition 2	u ^ 2 <= 1	T	F	-
		ACTIONS: SPECIFY A ROW FROM THE ACTION TABLE	1	1	2

Action Table

	DESCRIPTION	ACTION
1	Example action 1 called from D1 & D2 column in condition table	y = u;
2	Example action 2 called from D3 column in condition table	y = 1 / u;

150. Ábra – Truth Table belső felépítése – állapotok és események

A 150. Ábra bemutatja a Truth Table felépítését: a feltételek (conditions) és a döntések (decisions – jelölve D1-D3), melyek igaz és hamis (jelölve *T*, azaz True és *F*, azaz False) értékeket vehetnek fel, alkotják az alapot, míg az események (actions) határozzák meg a végrehajtandó műveleteket a megfelelő feltételek teljesülése esetén. A 150. Ábra által bemutatott táblázat az alábbi MATLAB kód segítségével írható le.

48. Forráskód – Truth Table struktúráját leíró programkód

```
function y = fcn(u)
aVarTruthTableCondition_1 = false;
aVarTruthTableCondition_2 = false;

% Example condition 1
aVarTruthTableCondition_1 = logical(u >= 0);
% Example condition 2

aVarTruthTableCondition_2 = logical(u^2 <= 1);

if (aVarTruthTableCondition_1 &&

  aVarTruthTableCondition_2)
elseif (~aVarTruthTableCondition_1 &&

    ~aVarTruthTableCondition_2)
else % Default


end

function aFcnTruthTableAction_1()
% Example action 1 called from D1 & D2 column in
% condition table
y = u;

function aFcnTruthTableAction_2()
% Example action 2 called from D3 column in condition
% table
y = 1 / u
```

Az igazságtábla-szerkesztő segítségével lehetőségünk van feltételek, műveletek és döntéshozatali beállítások szerkesztésére. A *Ports and Data Manager* segítségével módosíthatjuk a Stateflow adatokat és a portokat. Ezenkívül diagnosztikát futtathatunk le, amellyel észlelhetjük a hibákat. Nem utolsósorban lehetőségünk van a létrehozott tartalom megtekintésére a szimulációt követően. A Truth Table blokk alapértelmezetten két porttal rendelkezik. Jelölve: *u* – bemeneti, *y* – kimeneti portként szolgál.

Logikai kapuk modellezése

Következő példánkban az ismert logikai kapuk felépítését és működését mutatjuk be, melyekhez a True Table blokkokat fogjuk alkalmazni. A logikai kapuk igazságtáblázata határozza meg, hogy mely esetekben beszélhetünk IGAZ (True, 1), ill. HAMIS (False, 0) kimenetekről. Először ismertetjük az igazságtáblák típusait és egyes kimeneteit.

Jelölje A az első, B pedig a második állítást. A legalapvetőbb igazságtáblázatok a következők: AND, OR, XOR, NOT, NAND, NOR, XNOR.

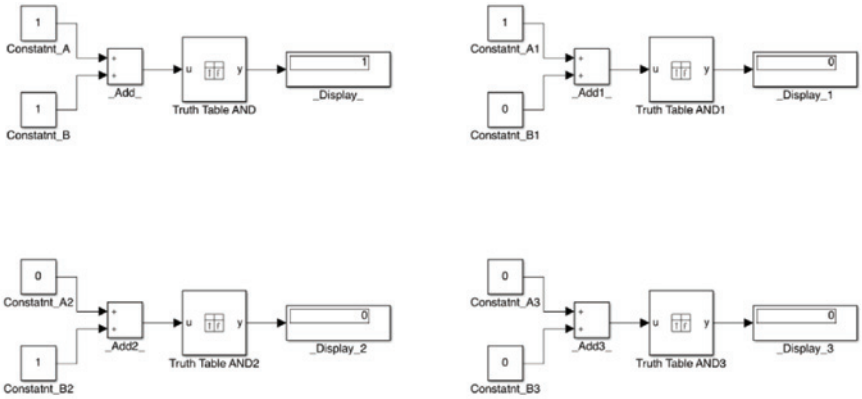
ÉS igazságtábla – AND

Az AND, másnéven ÉS kapu azzal a sajátossággal bír, hogy az adott bemeneti feltételekre csak akkor ad vissza igaz kimenetet, ha minden bemeneten szereplő feltétel igaz. Működését az alábbi táblázat írja le:

4. Táblázat – AND igazságtábla

A	B	AND - Output
1	1	1
1	0	0
0	1	0
0	0	0

A következőkben tekintsük meg e logikai kapu modelljének felépítését (151. Ábra) és működését (152. Ábra) mind a négy esetre.



151. Ábra – AND logikai kapu felépítése és kimenetei

Az egyes blokkok funkcióit az alábbiakban ismertetjük:

- **Constant:** A benne lévő értékek konstansként szolgálnak és a Truth Table bemenetét biztosítják az összeadó blokkon keresztül. Elérhetőség – Simulink.
- **Add:** Összegző blokk, amely példánk esetében a két konstans összegét továbbítja az igazságtáblának. Elérhetőség – Simulink.
- **Truth Table:** A logikai műveletek elvégzésére szolgál. A benne lévő feltételek alapján határozza meg a kimenetet. Elérhetőség – Stateflow.
- **Display:** Feladata a kimenet megjelenítése. Elérhetőség – Simulink.

Condition Table

	DESCRIPTION	CONDITION	D1	D2	D3
1	AND TRUE IF	u == 2	T	F	-
2	AND FALSE IF	(u == 1) || (u==0)	F	T	-
		ACTIONS: SPECIFY A ROW FROM THE ACTION TABLE	1	2	3

Action Table

	DESCRIPTION	ACTION
1	A = 1, B = 1	y = 1;
2	A = 1, B = 0 A = 0, B = 1 A = 0, B = 0	y = 0;
3	Other	y = -1;

152. Ábra – Truth Table logikai feltételei AND kapu esetén

Mivel két bemenettel dolgozunk (A és B állítások), és a Truth Table blokkunk általánosan egy *u* bemenettel rendelkezik, így a két konstans összegét vezetjük be az *u* bemenetre. A 3. Táblázat alapján kiindulva tudjuk, hogy az AND igazságtábla azon esetben ad vissza igaz értéket, ha mind a két állítás igaz. Jelen esetben két igaz állításra az összegző blokk 2-es értéket továbbít a bemenetre. A logikai műveleteket ez alapján, a következőképp állítottuk be: amennyiben az *u* értéke 2, akkor az *y* kimenet 1 (*Condition Table 1.* sora és Action Table 1. sora). Igaz és hamis, ill. két hamis állítás esetén a bemenetre továbbított érték

0, vagy 1. Ezen esetekben az *y* kimenet 0 értéket vesz fel (*Condition Table 2.* sora és *Action Table* 2. sora). Minden más esetben a kimenet -1, amely a nem megfelelő bemeneteket jelöli. Ahogyan azt a 152. Ábra is szemlélteti, a lényeges beállításokat a *Condition* és *Action* oszlopokban kell megadnunk. Láthatjuk, hogy a feltételek megadásakor alkalmazhatunk relációs, egyenlőségvizsgáló és összekötő operátorokat (==, <=, >=, ||, &&) egyaránt.

Állapotátmenet táblázat - State Transition Table

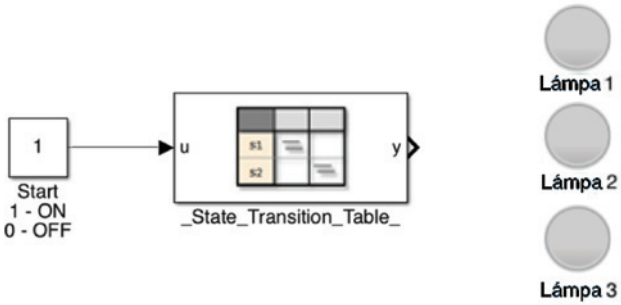
Fő funkciója a modális logika táblázatos formában való megjelenítése. Az állapotátmeneti táblázat blokk csak a MATLAB-ot használja műveletnyelvként. A State Transition Table szerkesztőjét szintén duplakattintással tudjuk előhívni. A szerkesztő segítségével lehetőségünk nyílik a modellünkhöz állapotokat és állapotműveleteket hozzáadni. Az állapotok között hierarchiát tudunk kialakítani. Ezen szerkesztő keretein belül áll módunkban meghatározni az állapotok közötti átmenet feltételeit. Az átmeneteknek három fajtáját határozhatjuk meg, és állíthatjuk be (alapértelmezett, belső és önciklus átmenetek). Ehhez a blokkhoz is lehetőségünk van bemeneti vagy kimeneti adatokat és eseményeket hozzárendelni. A hibák észleléséhez szintén módunkban áll diagnosztikát is lefuttatni. Ennél az esetnél lehetőségünk nyílik ún. törési pontok beállítására a hibakereséshez. A táblázat szerkesztése közben megtekinthető az automatikusan létrehozott tartalom.

Nézzünk meg egy példát a „State Transition Table" alkalmazására. Egy jelzőlámpa, vagy más néven szemafor, a mindennapokban történő közlekedést szabályzó eszköz, amely meggátolja az egyoldali hosszadalmas várakozási időt, amennyiben a közlekedési eszközök haladását például egy kereszteződésben sorbanállási modellekre vetítünk le. A jelzőlámpa modellje egy tipikus példa átmenetek közötti hierarchia kiépítésére és átmeneti feltételek definiálásának bemutatására. Következő példánkban egy ilyen modell elkészítésének lépéseit mutatjuk be.

Egy jelzőlámpa által irányított forgalom esetén általánosan három állapotot különböztethetünk meg:

- szabad közlekedés (jelölése zöld szín)
- tilos közlekedés – várakozás (jelölje piros szín)
- felkészülés, vagyis egy köztes átmenet (jelölése narancsszín)

Az alábbi ábrán tekintsük meg a modell felépítését, majd ismerjük meg az egyes blokkok funkcióit.



153. Ábra – Jelzőlámpa-modell felépítése

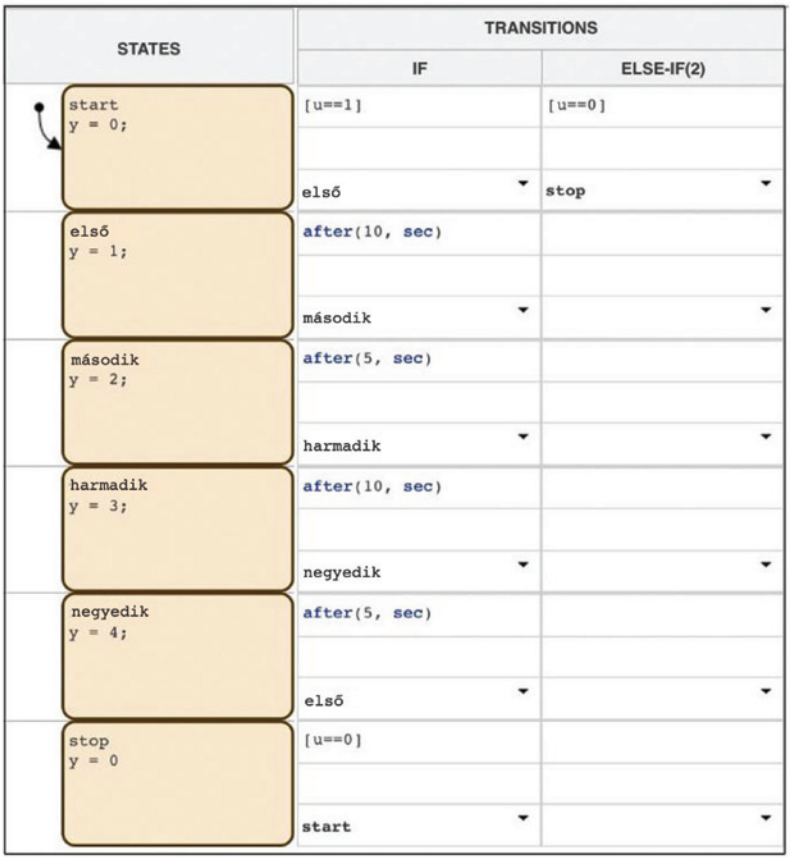
Az egyes blokkok funkcióit az alábbiakban ismertetjük:

- **Constant:** A benne lévő érték konstansként szolgál és a State Transition Table bemenetét biztosítja. Feladata a kezdő jel megadása, amely aktívvá, ill. inaktívvá teszi a jelzőlámpákat. Elérhetőség – Simulink.
- **State Transition Table:** Feladata az állapotok és azok hierarchiájának, átmenetfeltételeinek és állapotértékeinek vezérlése. Elérhetőség – Stateflow.
- **Lamp:** Feladatuk a kimenetek megjelenítése. Ahogyan azt a 153. Ábra is mutatja, a State Transition Table *y* kimenete nem áll vonal általi összeköttetésben a Lamp objektumokkal. Ez azzal magyarázható, hogy a Lamp képes összekötő vonalak nélkül kapcsolódni a modellben található más objektumokkal, amelyek kimenetei, mondhatni, wireles módon juttatják el az adatokat a Lamp-nek. A Lamp további beállításai megtekinthetők 155. Ábran. Elérhetőség – Simulink.

A grafikus kimenetek és az egyes blokkok beállításainak megtekintése előtt ismertetjük modellünk állapotait és jelöléseit:

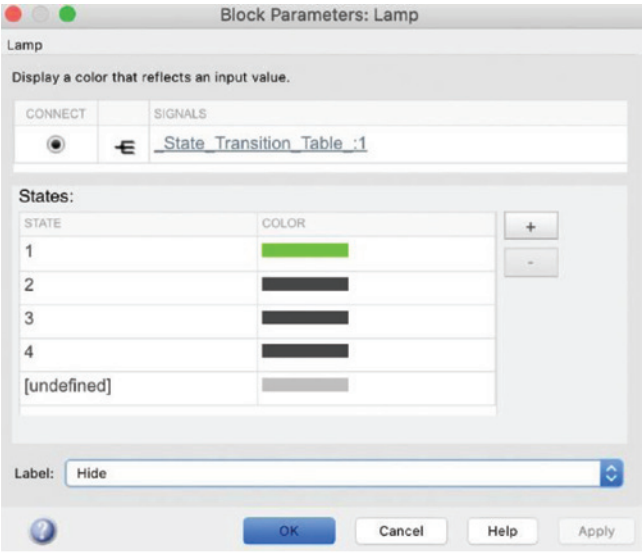
0. kezdeti állapot – **start**: megvizsgálja a bemeneti jelet (1 vagy 0)
1. állapot – **első**: amely az első jelzőlámpa (zöld) „világítását” idézi elő, azaz a szabad közlekedést (156. Ábra - a).
2. állapot – **második**: amely a második jelzőlámpa (sárga) „világítását” idézi elő, azaz a lassítást (156. Ábra - b).
3. állapot – **harmadik**: amely a harmadik jelzőlámpa (piros) „világítását” idézi elő, azaz a tilos közlekedést – várakozást (156. Ábra - c).
4. állapot – **negyedik**: amely a harmadik és második jelzőlámpa (piros és sárga) egyidejűleg történő „világítását” idézi elő, azaz a felkészülést (156. Ábra - d).
5. kikapcsolt állapot – **stop**: amely közvetlen a kezdő állapotból érhető el 0 bemeneti jel esetén.

A felsorolt állapotok beállításai megtekinthetők az alábbi ábrán:



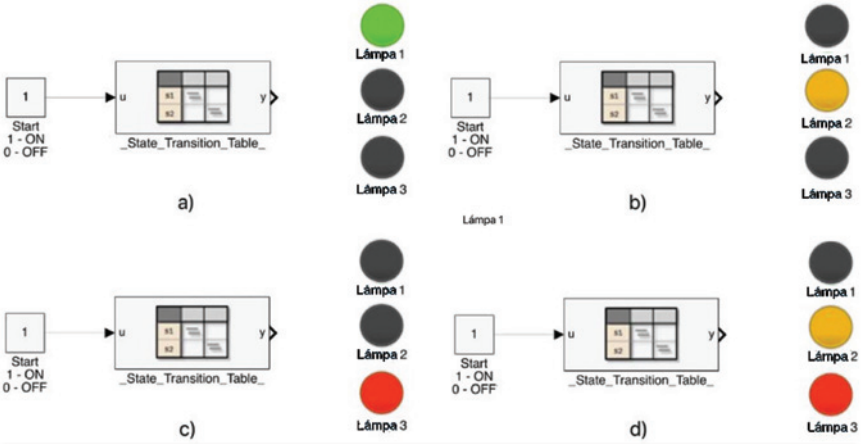
154. Ábra – A State Transition Table belső felépítése

Láthatjuk, hogy a kezdeti helyzettől eltekintve, öt különböző állapotról beszélhetünk. Ezen állapotok mindegyike esetében definiálnunk kell a várható grafikus kimeneteket. A 155. Ábra szemlélteti a Lamp objektum beállításait, ahol látható a vele kapcsolatban álló blokk (jelen esetben State Transition Table), valamint középen láthatóak a konkrét állapot értékek (1,...,4), amelyet az *y* kimenet határoz meg, illetve láthatók az egyes állapotokhoz tartozó kimeneti színek. Az 1. lámpa esetében tehát $y = 1$ kimenet során zöld színnel töltődik ki, más esetben feketével, ami a kikapcsolt állapotot demonstrálja.



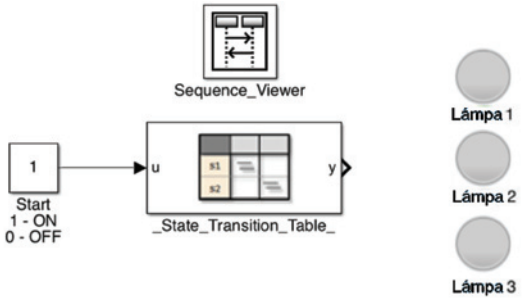
155. Ábra – Lamp blokk beállításai

Hasonló beállításokkal tudjuk szabályozni a többi Lamp blokk megfelelő működését is. Végül tekintsük meg a teljes szemafor modellünk kimeneteit.



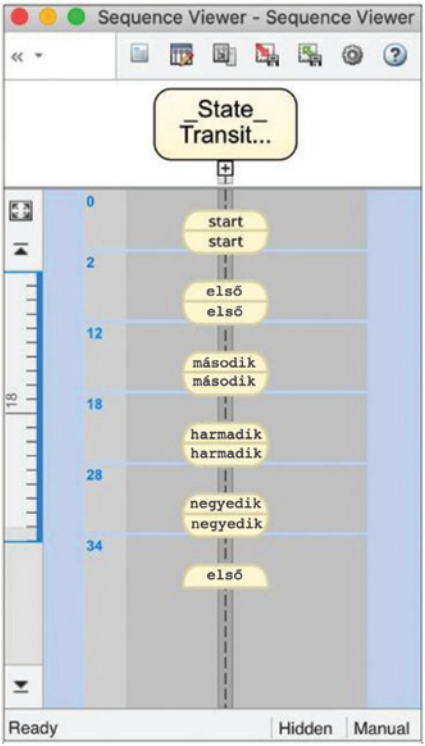
156. Ábra – Jelzőlámpa modell kimenetei – a): első, b): második, c): harmadik, d): negyedik állapotokat szemléltetik.

A meglévő modellünket egészítsük ki egy folyamatfigyelő egységgel. Ekkor a modellt alkotó blokkok a következőképpen ábrázolhatók.



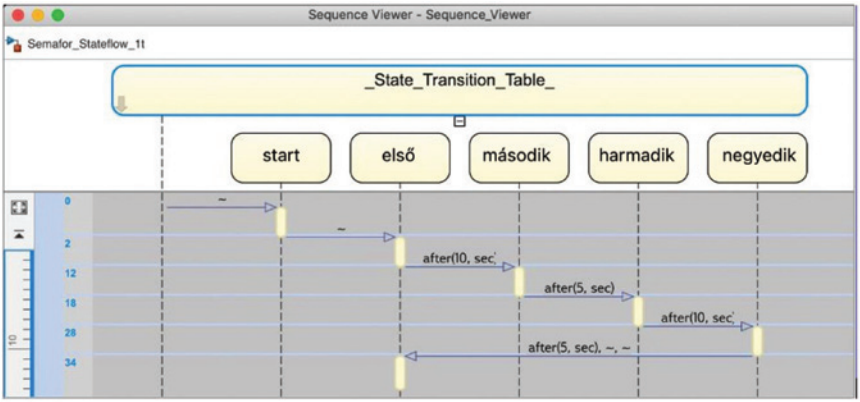
157. Ábra – Jelzőlámpa modell felépítése folyamat figyelővel.

A szimuláció lefuttatását követően, a folyamatfigyelőre duplán kattintva, az alábbi kimenetet kaptuk:



158. Ábra – Sequence Viewer kimenete

A 158. Ábra szemlélteti, hogy a lefutás alatt a folyamatfigyelő, milyen állapotátmeneteket észlelt és azt is, hogy az egyes állapotok mely blokkhoz tartoznak. Ezen modellünk esetében az állapotátmenet-táblázat és a benne meghatározott állapotok (melyeket már a fentiekben ismertettünk 154. Ábra) fedezhetők fel a Sequence Viewer kimenetén. Ezt a kimutatást lehetőségünk van részletesebben is megvizsgálni. A kiválasztott blokk alatt lévő "+" jelre kattintva áll módunkban a modulhoz tartozó részletes átmeneti statisztikákat megtekinteni. Jelen esetben a State Transition Table állapotátmenetei a Sequence Viewer-en belül az alábbiakban tekinthetők meg:



159. Ábra – Jelzőlámpa modell állapotátmeneteinek részleges áttekintése folyamatfigyelő segítségével.

Jól kivehető, hogy a 159. Ábra a jelzőlámpa egy teljes körét szimulálja. Első lépésként a bemeneti jelet ráküldi az első „start" állapotra. Ezt követően megvizsgálja a bemeneti jel értékét, amely esetünkben 1-es volt. A 154. Ábra mutatja be azon feltételeket, melyeknek megfelelően az 1-es bemeneti jelre válaszolva a modellünk átlép az első állapotba, amely a zöld lámpa világítását jelöli. 10 másodperc elteltével tovább halad a második állapotba, amely a narancssárga lámpa fényét jelöli. Az ezt követő 5. másodpercnél a modellünk tovább lép a harmadik állapotba, amely a Lámpa 3-at színezi pirosra. 10 másodperc után ismételt állapotváltozás történik, ahol a negyedik állapotba átlépve a piros fény mellett a narancssárga is felgyúl 5 másodperc erejéig. Majd végül ebből az állapotból tér vissza az első szakaszhoz.

Értelemszerűen amennyiben vissza szeretnénk lépni a részletes vizsgálatból, akkor ezt a választott blokk alatti „–" gombra kattintva tudjuk megtenni.

Folyamatfigyelő – Sequence Viewer

Feladata az üzenetek, események, állapotok, átmenetek és funkciók megjelenítése a blokkok között, a szimuláció során. A folyamatfigyelő egység ugyanis, képes a modellben szereplő átmeneteket automatikusan felismerni és elraktározni. Az átmeneti változásokat lehetőségünk van a szimuláció előrehaladtával folyamatosan lekövetni, de akár annak befejezését követően is áttekintést kaphatunk blokkokra lebontva, visszamenőlegesen az egyes állapotváltozásokról.

6.3.2. A MATLAB-ban található HyEQ Toolbox

Előző fejezeteinkben bemutattuk a Simulink, Simevents és a Stateflow könyvtárak által biztosított blokkok felhasználásával elkészített modelljeinket. A hibrid rendszerek kiépítésére viszont, adott egy MATLAB könyvtár, ahol új funkciókkal van lehetőségünk ezeket a modelleket felépíteni. A Hybrid Equation (HyEQ) Toolbox, a MATLAB/Simulink programban, hibrid dinamikus rendszerek szimulációjára készült. Ez az eszköztár képes bemenetekkel rendelkező egyedi, és összekapcsolt hibrid rendszerek szimulálására. Gyakorlati alkalmazását a korábbiakban bemutatott 5.3 fejezet f,g példáiban ismertettük. A Simulink implementáció négy alapvető paramétert tartalmaz, amelyek meghatározzák a hibrid rendszer dinamikáját. Ezek közé tartozik az áramlási térkép *(flow map)*, az áramlási halmaz *(flow set)*, az ugrási térkép *(jump map)* és az ugrási halmaz *(jump set)*, amelyeket a következőkben definiálunk. A (MathWorks, The MathWorks, Inc., 2021) alapján tekintsük meg egy H hibrid rendszer keretrendszerét egy $\mathbb{R}^n$ állapottéren, amelynek $\mathbb{R}^m$ bemeneti terét a következő objektumok határozzák meg:

- $C \subset \mathbb{R}^n \times \mathbb{R}^m$ halmaz jelöli az úgynevezett áramlási halmazt.
- $f \subset \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ függvény, amelyet áramlási térképnek nevezünk.
- $D \subset \mathbb{R}^n \times \mathbb{R}^m$ halmaz, amelyet ugrási halmaznak nevezünk.
- $g \subset \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ függvény, amelyet ugrási térképnek nevezünk.

A $H = (C, f, D, g)$ hibrid rendszerek MATLAB/Simulink szimulációját a következő módon írjuk fel:

$$H \begin{cases} \dot{x} = f(x, u) & x \in C \\ x^+ = g(x, u) & x \in D \end{cases} \tag{6.1}$$

ahol: $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$.

Az f áramlási térkép a folytonos dinamikát határozza meg a C áramlási halmazon, míg a g ugrási térkép a diszkrét dinamikát határozza meg a D ugrási halmazon.

A rendszer áramlásait és ugrásait integrátorrendszer számítja ki, amely olyan blokkokból áll, amelyek kiszámítják a hibrid rendszer folyamatos dinamikáját, ugrásokat váltanak ki, az ugrásoknál frissítik a rendszer állapotát és a szimulációs időt, valamint leállítják a szimulációt. Ezen kívül a HyEQ tartalmaz egy ún. lite szimulátort is, amely gyorsabb szimulációt tesz lehetővé a Simulink használata nélkül.

BEFEJEZÉS

Az MS területe az elmúlt évtizedekben jelentős fejlődésen ment keresztül, és ma már az élet számos területén elengedhetetlen eszköz. Jelen egyetemi jegyzet célja, hogy átfogó képet nyújtson az MS alapjairól, módszereiről és alkalmazási lehetőségeiről, különös tekintettel a MATLAB környezetre és annak eszköztárára, mint például a Simulink, Simevents és Stateflow könyvtárak. Az első fejezetekben megismerkedünk az MS alapfogalmaival, azok típusainak bemutatásával. A jegyzet különbséget tesz a diszkrét és folyamatos modellek között, részletezve, hogy mely szituációkban melyik típus alkalmazása célszerű. A modellezés során kiemeljük az adott szituációban fontos elemek kiválasztásának és a modell validálásának lépéseit. A tesztelési folyamat során a szimulációk jelentőségét hangsúlyozzuk, amelyek lehetővé teszik a modellek viselkedésének előzetes vizsgálatát költséghatékony és biztonságos módon. Az elméleti alapok után részletesen bemutatásra kerül a MATLAB környezetben történő modellezés és szimuláció. A Simulink könyvtár segítségével vizuálisan építhetünk fel modelleket, míg a Simevents és Stateflow könyvtárak speciális eszközkészleteket biztosítanak a különböző típusú rendszerek és események modellezéséhez. A gyakorlati példák révén az olvasó betekintést nyerhet abba, hogyan lehet komplex rendszereket modellezni és szimulálni, legyen szó mechanikai, elektromos vagy akár biológiai rendszerekről.

A könyv központi része az MS konkrét alkalmazási területeit tárgyalja. Részletesen foglalkozik a tervezési folyamatok optimalizálásával, a gyártási rendszerek hatékonyságának növelésével, valamint az oktatásban és kutatásban rejlő lehetőségekkel. Bemutatja, hogyan lehet a szimulációk segítségével költséges vagy veszélyes kísérletek helyett virtuálisan tesztelni új eljárásokat és termékeket, csökkentve ezzel a fejlesztési időt és költségeket. Kitekintést nyújt más tudományterületekre is, ahol az MS hasonlóan fontos szerepet játszik. Például az éghajlatmodellezésben, ahol a globális felmelegedés hatásait vizsgálják, vagy a pénzügyi szektorban, ahol a kockázatelemzés és előrejelzések készítése során használják a modelleket, vagy az orvostudományban, ahol a biológiai rendszerek szimulációja segíti a gyógyszerfejlesztést és a betegségek megértését.

Az MS jövőbeli irányai és kihívásai közül az egyik legjelentősebb fejlődési irány a mesterséges intelligencia (AI) és gépi tanulás (ML) integrálása a szimulációs rendszerekbe. Az AI és ML technológiák alkalmazása lehetővé teszi a prediktív modellezés új „dimenzióinak” feltárását, melyek révén pontosabb és dinamikusabb előrejelzések készíthetők.

A mesterséges intelligencia és gépi tanulás integrálása a szimulációs rendszerekbe számos új lehetőséget kínál. Az AI algoritmusok képesek nagy mennyiségű adat elemzésére és mintázatok felismerésére, amelyek a hagyományos módszerekkel nehezen lennének azonosíthatók. A gépi tanulás révén a modellek önállóan képesek tanulni és fejlődni a bevitt adatok alapján, ami növeli a szimulációk pontosságát és megbízhatóságát. A prediktív modellezés terén az AI és ML különösen hasznos lehet a dinamikus rendszerek elemzésében. Például az időjárás-előrejelzés, a pénzügyi piacok elemzése vagy a járványok terjedésének modellezése olyan komplex rendszerek, amelyekben a gépi tanulás jelentős előrelépést hozhat. Az AI-alapú szimulációk képesek adaptálódni a változó körülményekhez és valós idejű adatokat felhasználva finomítani a modelleket, ami jobb előrejelzéseket eredményez.

Az egyre növekvő számítási kapacitás és a felhőalapú rendszerek elterjedése szintén jelentős hatással van a szimulációs technológiák fejlődésére. A modern szuperszámítógépek és a felhőalapú számítástechnika lehetővé teszik a nagyléptékű és összetett szimulációk futtatását, amelyek korábban elképzelhetetlenek lettek volna. A felhőalapú rendszerek használata számos előnyt kínál. Először is, a felhőszolgáltatások rugalmas erőforrás-kezelést biztosítanak, így a felhasználók igény szerint skálázhatják a számítási kapacitást. Ez különösen fontos a nagy számítási igényű szimulációk esetében, ahol a hagyományos számítógépes erőforrások nem elegendők. Emellett a felhőalapú rendszerek hozzáférést biztosítanak a legmodernebb technológiákhoz és eszközökhöz, anélkül, hogy a felhasználóknak jelentős beruházásokat kellene tenniük hardver- és szoftverinfrastruktúrába.

Az MS jövője (ahogy múltja is) szorosan összefonódik az interdiszciplináris együttműködés erősítésével. A komplex rendszerek megértése és modellezése gyakran több tudományág együttes tudását és tapasztalatát igényli. Az együttműködés különböző szakterületek között, mint például a fizika, kémia, biológia, mérnöki tudományok és társadalomtudományok, új perspektívákat és megoldásokat hozhatnak.

Az MS jövője rendkívül ígéretes, tele van innovációs lehetőségekkel és kihívásokkal. A mesterséges intelligencia és gépi tanulás integrálása, az egyre növekvő számítási kapacitás és a felhőalapú rendszerek elterjedése mind hozzájárulnak a terület fejlődéséhez. Az olvasó betekintést nyerhetett a terület elméleti alapjaiba, gyakorlati alkalmazásaiba, és jövőbeli lehetőségeibe, így képes lesz ezeket az ismereteket saját szakmai tevékenységében hatékonyan alkalmazni. Az MS folyamatos fejlődése révén mindig lesz új felfedezni- és tanulnivaló, amely hozzájárulhat a tudomány és technológia előrehaladásához.

IRODALOMJEGYZÉK

1. Ábrahám, P. D. (2019). A Toolbox for the Reachability Analysis of Hybrid Systems using Geometric Approximations (HyPro). Letöltés dátuma: 2024. 06 01, forrás: RWTH Aachen University: https://ths.rwth-aachen.de/research/projects/hypro/bouncing-ball/
2. Alberto, B. (2003). Modeling, Control, and Reachability Analysis of Discrete-Time Hybrid Systems. Siena: DISC School.
3. Alur, R., Courcoubetis, C., Halbawchs, N., Henzinger, T., Ho, P., X. Nicollin, A. O., . . . Yovine, S. (1995). The Algorithmic Analysis of Hybrid Systems. Theoretical Computer Science, 138.
4. Ashish, A. (2006). Logical Modeling Frameworks for the Optimization of Discrete-Continuous Systems - PhD Dissertation. Pittsburgh: Carneige Mellon University.
5. B. Dabney, J., & L. Harman, T. (2004). Mastering Simulink. Pearson Prentice Hall.
6. Bail, J. L., Alla, H., & David, R. (1991). Hybrid Petri Nets. European Control Conference, (old.: 1471-1477). Grenoble.
7. Barros, J. F. (1997. October 1). Modeling Formalisms for Dynamic Structure Systems. ACM Transactions on Modeling and Computer Simulation, Volume 7, Issue 4, old.: 501-515.
8. Bartha, T., Majzik, I., & Pataricza, A. (2021). Petri hálók: Alapelemek és kiterjesztések. Letöltés dátuma: 2024. 06 01, forrás: inf.mit.bme.hu: https://inf.mit.bme.hu/sites/default/files/materials/category/kategória/oktatás/msc-tárgyak/formális-módszerek/16/FM-2016_EA07a_PN_alapelemek_kiterjesztesek.pdf
9. Beek, B. v., Jansen, N. G., Rooda, K. E., Schiffelers, R. R., Man, K. L., & Reniers, M. A. (2003). Relating Chi to Hybrid Automata. Proceedings of the 2003 Winter Simulation Conference.
10. Brauer, F., & Castillo-Chavez, C. (2012). Mathematical Models in Population Biology and Epidemiology (Second Edition). Springer.
11. Chaturvedi, D. K. (2017). Modeling and Simulation of Systems Using MATLAB and Simulink. CRC Press.
12. Csikó, C. (2015). Rekurzív sorozatok. Budapest: Eötvös Loránd Tudományegyetem.
13. Dabney, J. B., & Harman, T. L. (2004). Mastering Simulink. Pearson.
14. Drogňa, J. (2012). Efficient Modeling of Hybrid Systems - Diploma Thesis. Bratislava: Slovak University of Technology in Bratislava.
15. Fehér, Z., & Jaruska, L. (2015). Valószínűségszámítás és statisztika alapjai (első. kiad.). Komráno: Univerzita J. Selyeho.

16. Frost, S. (2018). SIR model. Letöltés dátuma: 2024. 06 01, forrás: epirecipes - Developing a cookbook of epidemiological models: http://epirecip.es/epicookbook/chapters/sir/intro
17. Fewster, R. (2014). COURSE NOTES STATS 325 Stochastic Processes. University of Auckland.
18. Gyenge, Á. (2010). A Lorenz-attraktor. Letöltés dátuma: 2021. 10 12, forrás: https://adamgyenge.gitlab.io/docs/notes/lorentz.pdf
19. Hangos, K., & Szederkényi, G. (1999). Dinamikus rendszerek paramétereinek becslése. Veszprém: Veszprémi Egyetemi Kiadó.
20. Harel, D. (1987). Statecharts: A Visual Formalism for Complex Systems. Sci. Comp. Prog., 231-274.
21. Inc., MathWorks. (2021). MATLAB & SIMULATION - User's Guide. Letöltés dátuma: 2024. 06 01, forrás: https://www.mathworks.com/help/pdf_doc/matlab/learn_matlab.pdf
22. Johanson, K. H., Lygeros, J., & Sastry, S. (2004). Modeling of Hybrid Systems. CONTROL SYSTEMS, ROBOTICS AND AUTOMATION – Vol. XV.
23. Kermack, W. O., & McKendrick, A. G. (1921). A contribution to the mathematical theory of epidemics. Proceedings of the The Royal Society A, 22. doi:https://doi.org/10.1098/rspa.1927.0118
24. Kmeť, T. (2018). Modellezés és Szimuláció 2020. Letöltés dátuma: 2024. 06 01, forrás: https://e-learning.ujs.sk/course/view.php?id=391
25. Kmeť, T. (2021). Modellezés és szimuláció elmélete és eszközei. Letöltés dátuma: 2024. 06 01, forrás: https://e-learning.ujs.sk/course/view.php?id=392
26. Ljung, L. (2001). Black-box Models from Input-output Measurements. IEEE Instrumentation and Measurement Technology Conference. Budapest.
27. Lunze, J., & Lamnabhi-Lagarrigue, F. (2009). Handbook of Hybrid Systems Control: Theory, Tools, Applications. Cambridge: Cambridge University Press.
28. Lygeros, J., Sastr, S., & Tomlin, C. (2012). Hybrid Systems: Foundations, advanced topics and applications.
29. MathWorks. (2021). The MathWorks, Inc. Letöltés dátuma: 2024. 06 01, forrás: https://www.mathworks.com
30. MathWorks. (2022). The MathWorks Inc. Letöltés dátuma: 2024. 06 01, forrás: MathWorks Help Center: https://www.mathworks.com/help/simevents/
31. Nagy, G. V. (2016). Fibonacci-számok és Catalan számok. Szeged, Szeged: SZTE Bolyai Intézet, Hungary.
32. Packard, N. H., & Wolfram, S. (1985). Two-Dimensional Cellular Automata. Journal of Statistical Physics, 38(5/6), 901-946.

33. Pokorádi, L. (2013). Rendszertechnika. Budapest: TERC Kereskedelmi és Szolgáltató Kft.
34. Proß, S., & Bachmann, B. (2011). An Advanced Environment for Hybrid Modeling of Biological Systems Based on Modelica. Journal of Integrative Bioinformatics, 8(1):152.
35. Rudolf, Á. (2012). Komplex rendszerek szimulációs módszerei. Letöltés dátuma: 2024. 06 01, forrás: http://adamrudolf.web.elte.hu/letoltesek/jegyzokonyvek/MSc_2.felev_komplex_rendszerek_szimulacios_modszerei/Komplex02_Rudolf.pdf
36. Sahbani, A., & Pascal, J. - C. (2000). Simulation of hybrid systems using stateflow. CiteSeerX, 1-2.
37. Sanfelice, R. G., Nanez, P., & Copp, D. A. (2013). HyEQ: A Toolbox for Simulation of Hybrid Dynamical Systems. Letöltés dátuma: 2024. 06 01, forrás: https://www.mathworks.com/videos/hyeq-a-toolbox-for-simulation-of-hybrid-dynamical-systems-81992.html
38. Schaft, A. v., & Schumacher, H. (2000). An Introduction to Hybrid Dynamical Systems. Amsterdam: Springer.
39. Szeidl, L. (2009). Tömegkiszolgálás. Budapest: Óbudai Egyetem - Neumann János Informatikai Kar. Forrás: https://web.archive.org/web/20170519074928/http://uni-obuda.hu/users/szeidll/Osszefoglalo.pdf
40. Sztrik, J. (2009). A sorbanállási elmélet alapjai. Debrecen: Debreceni Egyetem.
41. Weisstein, E. W. (2022). Elementary Cellular Automaton. Letöltés dátuma: 2024. 06 01, forrás: MathWorld - A Wolfram Web Resource: https://mathworld.wolfram.com/ElementaryCellularAutomaton.html
42. Weisstein, E. W. (2022). Game of Life. Letöltés dátuma: 2024. 06 01, forrás: MathWorld - A Wolfram Web Resource: https://mathworld.wolfram.com/GameofLife.html
43. Wojtowitcz, M. (2006). Cellular Automata Rules Lexicon. Letöltés dátuma: 2024. 06 01, forrás: http://www.mirekw.com/ca/rullex_gene.html
44. Zeigler, B. P., Muzy, A., & Kofman, E. (2019). Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations (3rd. ed.). USA: Academic Press, Inc.
45. Zhang, W., Branicky, S. M., & Philips, S. M. (2001). Stability of networked control systems. IEEE Control Systems Magazine, old.: 84-99.
46. Kmeť, T. (2005). Mathematical Modelling and Simulation of Biological Systems. Nitra: AM Press Nitra.

FORRÁSKÓDJEGYZÉK

Valamennyi bemutatott modell forráskódja megtalálható az alábbi linken:
https://github.com/JSelyeUniversity/folyamatmodellezes

ÁBRAJEGYZÉK:

Autori/Szerzők:
prof. RNDr. Kmeť Tibor, CSc.
Mgr. Annuš Norbert, PhD.
PaedDr. Csóka Márk, PhD.
Mgr. Paksi Dávid, PhD.

Názov/Cím:
Folyamatmodellezés és számítógépes szimuláció a MATLAB-ban

Recenzenti/recenzensek:
doc. RNDr. Bukor József, PhD.
Dr. habil. Kiss Attila Elemér, CSc.

Jazyková úprava / Nyelvi lektorálás:
Mgr. Bugár S. Veronika, PhD.

Vydavateľ/Kiadó:
Univerzita J. Selyeho
Fakulta ekonómie a informatiky
Bratislavská cesta 3322
SK-945 01 Komárno
www.ujs.sk

Tlačiareň/Nyomda:
DMC s.r.o.

Rozsah/Terjedelem:
7 AH/szerzői ív

Počet výtlačkov/Példányszám:
100

Rok vydania/Megjelenés éve
2024

Prvé vydanie/Első kiadás:

ISBN 978-80-8122-503-1