Budapesti Közgazdaságtudományi és Államigazgatási Egyetem
Matematikai Közgazdaságtan és Ökonometria Tanszék

EVOLÚCIÓS GAZDASÁGOK SZIMULÁCIÓJA

Ph.D. értekezés

Benedek Gábor

Budapest, 2003

Zolának

Tartalomjegyzék

„A tudomány nem próbál végső magyarázatot adni, fogalmakat értelmezni is alig. A természettudomány modelleket alkot. Modell alatt egy olyan matematikai struktúra értendő, amelyik – bizonyos szóbeli interpretáció hozzáfűzésével – leírja a jelenséget. Egy ilyen matematikai struktúra létjogosultságát egyedül az adja, hogy sikeresen előrelátja a jelenségeket, tehát működik."

(Neumann János)

1. Fejezet

Bevezetés

A modern gazdaságelméletben alkalmazott módszerek és gazdasági elemzések eszköztára a 20. század végére rendkívüli módon kiszélesedett. Leginkább természetesen a matematikai és statisztikai módszerek épültek be és fejlődtek tovább közgazdasági modelleken keresztül. Így napjainkban kevés olyan elméleti publikáció jelenik meg, amely ne alkalmazná a *játékelmélet*, a *variációszámítás*, a *sztochasztikus folyamatok* vagy más bonyolult területek módszereit. Az egzakt módszerek bonyolultságának növekedése azonban sajátos ellentmondást szült az elmélet és a gyakorlati alkalmazások között. A modern matematikai és statisztikai módszerek ugyanis pontosan azt a célt hivatottak szolgálni, hogy a gazdaság folyamatait jobban és pontosabban magyarázzák meg, és mind a gazdaságpolitikai (makroszintű), mind a vállalati (mikro szintű) döntéshozók számára olyan apparátust biztosítsanak, amelynek segítségével megalapozottabb és biztosabb döntéseket hozhatnak. A valóságban azonban az egyre bővülő szakmai háttérismereteket igénylő módszereket és a nehezen interpretálható modelleket a gazdaság gyakorlati szakemberei egyre nehezebben képesek adaptálni. Az elméleti lehetőség és a gyakorlati oldal egymástól erősen eltávolodott. Kivételként a pénzügyi területen érezhetjük azt, hogy a modern matematikai módszereket nap mint nap alkalmazzák (többek között a Wall Street matematikus- és fizikus-közgazdász tanácsadói), a valóságban azonban a legtöbb napi szintű elemzés és döntés (pl. kockázatkezelés) 30 éves módszerekre épül.

A közgazdaságtan másik módszertani forrása a számítástechnika alkalmazása. A számítógép megjelenése a legtöbb tudományágat döntő mértékben befolyásolta, pedig tulajdonképpen csupán két funkciót valósít meg; egyrészt rendkívül nagy méretű adatbázist képes hatékonyan tárolni és kezelni, másrészt óriási sebességgel képes számolni. E tulajdonságok miatt lehetővé vált, hogy a nagy mennyiségű információ birtokában más modellek épüljenek, illetve az elméleteket és az előrejelzéseket úgy

változtassák, hogy azok a megfigyelt és rögzített adatok tükrében jobban és pontosabban írják le a valóságot. A számítógépes modellezés folytán lehetőség nyílt olyan korlátozó feltételezések elhagyására, amelyek egy-egy korábbi modellt irreálissá tettek. (Ezek közül az egyik legsúlyosabb feltételezés a linearitás). Így az *analitikus* megoldások mellett megjelentek a *numerikus* megoldások. A számítógépes modellezés megjelenésével ugyanakkor azt várhatnánk, hogy a közgazdaság elméleti, illetve gyakorlati területén dolgozó szakemberek közötti távolság csökken, hiszen az elméleti szakemberek jóval komplexebb modelleket is ki tudnak értékelni, a gazdasági döntéshozók pedig jobban működő modellek eredményeire alapozva hozhatják meg döntéseiket. Sajnos a közgazdaságtudomány területén ez a közeledés még csak kezdeti stádiumban van. A számítógép töretlen fejlődésének és egyre népszerűbbé válásának következtében azonban könnyen lehet, hogy a közeli jövőben a számítástechnika fogja betölteni a híd szerepét az elméleti és a gyakorlati oldal között.

Számtalan esetben alkalmaznak analitikusan meg nem határozható – vagy nehezen meghatározható – problémák kezelésére numerikus módszereket a közgazdászok. Ilyen például a numerikus integrálás, a numerikus egyenlet/egyenlőtlenség megoldás, a differenciálegyenletek numerikus megoldása vagy az operációkutatás. Sőt, sokszor bizonyos problémák analitikus megoldására is numerikus módszereket vesznek igénybe, mint például a *szimbolikus programozás*[1] (integrálás, algebrai egyenlet megoldás, differenciálegyenlet megoldás, stb.) esetén.

A jelen értekezés azonban nem általánosságban a numerikus módszerekkel, illetve ezek közgazdasági alkalmazásával, hanem ennek egy részterületével, a *szimulációval* foglalkozik. A vizsgálódás középpontjában minden esetben valamilyen gazdasági modell van, amelynek vizsgálatára – nagyon sok esetben az analitikus módszerek mellett – numerikus módszereket használunk. Tekintsük ezt a közgazdasági modellt egy olyan leképzésnek, amely a valóságban megfigyelhető változók halmazához rendel egy halmazt. Ilyen értelemben a szimuláció nem más, mint ezen leképzések (bizonyos esetben akár függvények) analízise. Így a szimuláció legfontosabb feladatai gyakorlatilag megegyeznek az analízisben meghatározott feladatokkal, úgy mint függvénykiértékelés (*leszámolás*), egyenletmegoldás (*célérték-keresés*) vagy szélső-érték-keresés.

Kiinduló esetben a gazdasági modell elegendően egyszerű ahhoz, hogy analitikus tanulmányozása lehetséges legyen. A szimulációra abban az esetben van szükség, amikor az egyes feltételezések feloldása, illetve a modell dimenziójának növelése annyira megnehezíti a feladatot, hogy annak „kézi” kiszámolása lehetetlenné válik. Még

[1] Szimbolikus programozás alatt olyan matematikai műveletek számítógépes megoldását értjük, ahol a matematikai kifejezésben paraméterek is szerepelnek (ld.: Maple vagy Matlab).

ilyen esetekben is jellemző az, hogy az analitikusan kiszámolható eredményeket – összehasonlítás végett – numerikusan is előállítják.[2]

Az értekezésben azt a célt tűztük ki, hogy bemutassuk, hogyan lehet a szimulációs módszertant hatékonyan alkalmazni olyan esetekben, amikor átléptük az analitikus módszerek határait. Maga a szimulációs módszertan is rendkívül fiatal, kevesebb, mint ötven éves. Az elméleti közgazdaságtanban való elterjedéséhez azonban az igazi lökést a mesterséges intelligencia kutatások – elsősorban 1990-es éveinek – eredményei adták. A társadalomtudományok számára felhasználható, mesterséges intelligenciát is alkalmazó, szimulációs módszertan csak 2001-ben (!) a *Journal of Economic Dynamics and Control* külön számában kapta meg az azt megillető publicitást és önálló nevet – *Agent-based computational economics* (ACE)[3] –, amelyet *sokszereplős szimulációs közgazdaságtannak* fordítunk a továbbiakban. Az értekezés az általánosan alkalmazható szimulációs megoldásokon túlmenően erre a fiatal és rendkívül ígéretesnek látszó módszertani területre kíván fókuszálni.

1.1. Az értekezés felépítése

Az értekezés két jól elkülöníthető részből áll.

Az első rész – 2., 3., és 4., fejezet – a módszertani ismertetést tartalmazza, és az a célja, hogy bevezesse azokat a fogalmakat és eljárásokat, amelyeket a gyakorlati szimulációs modellek esetében alkalmazni lehet. Ezek a fejezetek erősen épülnek a szimulációs szakirodalomra, valamint a szerző által látogatott *manchesteri* és *limericki* egyetemeken oktatott szimulációs kurzusok jegyzeteire. A fejezetek strukturálása szempontjából a szerző által a *Budapesti Közgazdaságtudományi és Államigazgatási Egyetemen* oktatott *Szimuláció* c. tárgy jelentett nagy segítséget. A módszertani fejezet során számos példa illusztrálja a fogalmakat, amelyeket a szakirodalomban fellelhető modellek mellett a szerző társszerzős és saját publikációi, valamint hazai vállalatoknál végzett kutatási munkái képeznek.

A második rész – 5., 6., és 7., fejezet – a szimulációs módszertan konkrét alkalmazását tartalmazza. Ezek a modellek bár látszólag elkülönülnek, a második rész végére egy új kutatási irányt határoznak meg, amely mind az elméleti, mind a gyakorlati közgazdászok számára alkalmazható. A fejezetekben szereplő modellek részben a szerző publikációi, illetve konferencia előadásai, részben pedig további kutatási eredményei.

[2] A szakirodalom ezt a technikát *benchmarking*-nak nevezik.
[3] Tesfatsion [2001]

Az értekezés első része a szimulációs módszertan bemutatásával kezdődik (2. fejezet). A szakirodalomban számos olyan könyv található, amely a szimulációs módszerek alkalmazását írja le. A probléma azonban az, hogy ezek elsősorban a műszaki tudományok területére koncentrálnak, és az itt szereplő közgazdasági modellek szinte kizárólag vállalatgazdasági kérdésekkel foglalkoznak. (Például kiszolgálási rendszerek szimulációjával, készletgazdálkodási problémával, stb.) A 2. fejezetben ezekre az irodalmakra építünk, de az elméleti közgazdaságtani és pénzügyi alkalmazásokra fókuszálunk.

A 3. fejezetben a mesterséges intelligencia módszereit mutatjuk be. Ez a fejezet erősen épít Benedek [2000, 2001] publikációira, de azokat kiegészíti a lényeges algoritmikus részekkel és tételekkel.

A 2. és 3. fejezet előkészíti a 4. fejezetben bemutatásra kerülő *sokszereplős szimulációs közgazdaságtan* módszerét, továbbiakban ACEt. Ez a fejezet kapcsolja össze a szimulációs és mesterséges intelligencia módszerek lehetőségeit és alkalmazza olyan közgazdasági modellekben, ahol elkülönült szereplők viselkedése határoz meg valamilyen gazdasági viselkedést vagy jelenséget. A második fejezet modelljei erre a módszertanra építkeznek. A fejezet elsősorban a *Journal of Economic Dynamics and Control* ACE módszertannal foglalkozó cikkeire és a Leigh Tesfatsion által szerkesztett ACE hivatalos honlapjáról letölthető anyagokra épít.[4]

Az 5. fejezetben a Kiyotaki-Wright modellel foglalkozunk. (Kiyotaki, Wright [1989]). Az eredeti modell ismertetése után olyan kibővítésekkel foglalkozunk, amelynek analitikus megoldása ismeretlen, vagy az analitikus megoldást csak jóval később, a numerikus eredmények inspirálására sikerült meghatározni.

A 6. fejezetben egy látszólag új témakört érintünk, a társadalmi hálózatok problémáját. A modellt az információ telekommunikációs eszközökön keresztüli terjedésének a kutatása és szimulációja alapozta meg. A kutatás eredményeinek ismertetése mellett a fejezetben – elsősorban Watts [1999] munkája segítségével – bevezetünk néhány alapvető gráfelméleti fogalmat.

A 7. fejezetben megmutatjuk azt, hogy hogyan illeszkedik bele a Kiyotaki-Wright-féle modell világába a társadalmi hálózatok problémaköre. Sőt továbbmegyünk, és felvillantunk néhány olyan elméleti modellt, ahol az ACE módszertan és a hálózati szemlélet alkalmazásával új eredmények érhetők el.

A 8. fejezetben összefoglaljuk az eredményeket.

[4] http://www.econ.iastate.edu/tesfatsi/ace

1.2. Saját eredmények

A magyar közgazdasági szakirodalomban rendkívül kevés szimulációval foglalkozó tanulmány található. Ezek nagy része is valamilyen műszaki vagy vállalatgazdasági problémához kapcsolódnak. A kifejezetten elméleti kutatásokban a szimuláció alkalmazását kizárólag statisztikai/ökonometriai, illetve pénzügyi kutatásokban tapasztalhatjuk. Így a módszertan elméleti kutatásokba történő bevezetése a szerző önálló munkája. Különösen igaz ez az ACE módszertanra, amely annyira fiatal terület, hogy hazai publikációk még nem születtek ebben a témában.

Az 5. fejezetben bemutatásra kerülő Kiyotaki-Wright modell kibővítése illeszkedik a nemzetközi szakirodalomhoz. Az itt megfogalmazott eredmények újdonsága elsősorban az új módszerek bevezetésének és alkalmazhatóságának vizsgálatában rejlik. A szakirodalomban szereplő eredmények numerikus módszerekkel történő reprodukálása biztosítja a komplexebb kiterjesztések vizsgálatának lehetőségét.

A 6. fejezet modellje és az ott alkalmazott módszerek kifejlesztése a szerző munkája. (Természetesen a vállalati kutatás kivitelezésén egy egész csapat dolgozott). A 7. fejezet modelljei szorosan kapcsolódnak az előző fejezetekhez és szintén saját eredménynek tekinthetők.

2. Fejezet

Szimulációs módszertan

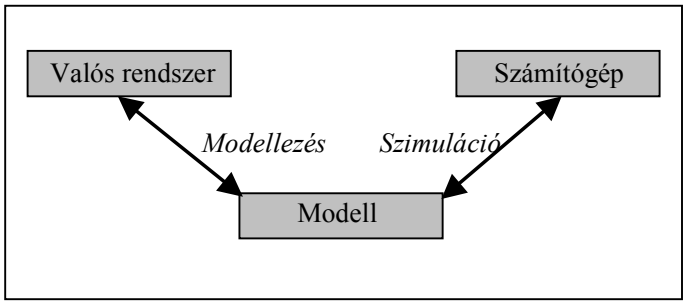
„Ahhoz, hogy a szimulációs módszerek magas szintű alkalmazása elterjedjen a társadalomtudományok területén, el kell fogadtatni, hogy az induktív és a deduktív módszerekkel szemben a tudomány művelésének harmadik útja a szimulációs kísérletezés." Az idézet Axelrod [1997] cikkéből származik, amelyben a társadalomtudományokban alkalmazott szimulációs módszerek jelenét és jövőjét vizsgálta. A cikkből három gondolatot emelünk ki. Az első, hogy a szimulációs modellek publikálása során nem elegendő az eredmények és a modell ismertetése. Természetesen helykorlátok miatt nincs lehetőség a forráskód publikálására, azonban a CD-k és az Internet segítségével lehetőséget kell biztosítani arra, hogy minden érdeklődő személyesen végrehajthassa (*futtathassa*) a kísérleteket. A második fontos gondolat, hogy a területen tevékenykedő szakemberek a korábban publikált szimulációkat újra végrehajtsák, kezdve a modell építésétől, a programozáson átmenve, a futtatásokig. Erre azért van nagy szükség, mert a numerikus módszerek sok esetben nem bizonyító erejűek. Ahhoz, hogy a hibákat és az igazi kihívásokat megtaláljuk, sokszor szükséges mások kutatási eredményeit a legelső lépéstől kezdve felépíteni. Végezetül Axelrod azt hangsúlyozza, hogy a szimulációs módszertan elfogadtatásához arra is szükség van, hogy az előzőek nyomán megalakuljon a társadalomtudósok olyan társasága, amely szimulációs módszerek segítségével végzi kutatásait.

Az értekezés során ezekre a gondolatokra külön nem hívjuk fel a figyelmet. Mivel azonban a módszertan önállósodása szempontjából rendkívül fontosnak tartjuk, itt a bevezetőben hangsúlyoztuk.

2.1. Definíció, alapfogalmak

A szimulációt nehéz definiálni. Az egyik lehetséges definíció szerint „a szimuláció egy rendszer modelljének a megfelelő bemenetekkel (*inputokkal*) történő ellátása, működtetése (*driving*) és a kimenetek (*outputok*) megfigyelése". (Bratley et al. [1987]). A szimulációs módszertan közgazdasági alkalmazása tehát a következőképpen képzelhető el. Megfigyeljük a valós rendszert, majd azt megkíséreljük – legfontosabb tulajdonságaik alapján – formálisan leírni. Ezt nevezzük *modellező tevékenységnek*. Ha a modellt *beprogramozzuk*, úgy a valós rendszer egy *virtuális reprodukcióját* kapjuk. Ebben azután – bizonyos paramétereket megváltoztatva – a valós rendszer viselkedésére vonatkozó megfigyelések érdekében *kísérleteket* hajthatunk végre. A programozási és kísérletezési tevékenységet nevezzük *szimulációs tevékenységnek*. (Ld.: 2.1. ábra).

Felmerül a kérdés, hogy mikor érdemes szimulációt alkalmazni és milyen célokat lehet ezzel elérni.



2.1. ábra: Szimuláció definíciója

A „mikor" kérdésre viszonylag egyszerű a válasz. Akkor kell szimulációt alkalmazni, amikor minden más lehetőség *költségesebb*. A más lehetőségen általában két alternatívát szoktak érteni. Az első az analitikus megoldás, a másik a valós rendszeren végzett kísérletezés. Mindegyik esetben a költség is igen összetett fogalom. Költségként jelenik meg az időráfordítás, az emberi erőforrás, az anyagi ráfordítás és a kockázati tényező. A valós kísérletezéssel szemben a modellezés legfőbb előnyének azt szokták megemlíteni, hogy valós beavatkozás esetén rendkívül sok idő telik el addig, amíg az adott beavatkozás hatásai mérhetővé válnak. Továbbá, nagy volumenű befektetéseket igénylő döntés esetében nem lehet „próbálgatni". A modellezésnek ezzel szemben az a hátránya, hogy – mivel a valóság egy leegyszerűsített mását valósítja meg, – elképzelhető a kiszámított értékek és a tényleges realizáció (valós rendszer eredményeinek) különbözősége.

Természetesnek tűnik az is, hogy abban az esetben, amikor analitikus megoldás áll rendelkezésre, felesleges a szimuláció. Sok esetben azonban az analitikus megoldáshoz vezető út nem egyszerű. Ilyen esetekben rendkívül nagy segítséget nyújthat a szimuláció, elsősorban azért, hogy pontosabb képet alkothassunk a feladatról. Előfordulhat az is, hogy egy problémának elvileg meg lehetne határozni az egzakt, analitikus megoldását, ennek ellenére szimulációs módszert használunk, mivel a probléma analitikus megoldása annyira bonyolult – és mechanikus –, hogy egyszerűbb a numerikus vizsgálat eredményeit használni. (Például *leszámlálás*).

A szimulációs módszerek felhasználási területei a következők:[1] *előrejelzés*, *feladat-megoldás*, *gyakorlatszerzés*, *szórakozás*, *oktatás*, *bizonyítás* és *kutatás*.

[1] Többek között Axelrod [1997] alapján.

1. *Előrejelzés*. Ezt a célt szolgálja a vállalatgazdasági alkalmazások nagy része. Például egy vállalat arra kíváncsi, hogy különböző árazás mellett várhatóan mekkora értékesítésre számíthat a jövőben.

2. *Feladatmegoldás*. A szimuláció arra is alkalmas, hogy különböző – más módon nem meghatározható – feladatokat oldjunk meg a segítségével. Ez a terület tipikusan a mesterséges intelligencia kutatások kapcsolódási pontja. Olyan feladatokat lehet megoldani, mint például az automatikus orvosi diagnóziskészítés (*adatbányászattal*), beszédfelismerés (*neurális hálózatokkal*), vagy függvényoptimalizálás (*genetikus algoritmussal*).

3. *Gyakorlatszerzés*. Az első szimulációk elsősorban azt a célt szolgálják, hogy a szimulációt felhasználó emberek képességét javítsák egy, a valóságot kellő pontossággal leíró, interaktív környezetben. Klasszikus példái ennek az autó és repülő szimulátorok.

4. *Szórakozás*. Csak egy lépés az előző ponttól és máris kitalált környezetek sokaságát teremthetjük meg erre a célra.

5. *Oktatás*. Az oktatásban használt szimulációs modellek legfontosabb célja, hogy lehetővé tegyék a hallgatók számára a valóságban meghúzódó törvényszerűségek és összefüggések megismerését. Jó példa a Közgazdasági Egyetemen is oktatott *LUDUS* vállalati szimulációs tantárgy.

6. *Bizonyítás*. Ez a leginkább vitatott terület, annak ellenére, hogy egy klasszikus és népszerű matematikai problémát, a négy-szín tételt, számítógépes (szimulációs) lépések segítségével oldották meg. A szimulációs eszközök segítségével történő bizonyítás olyan esetekben a leggyakoribb, amikor véges számú variációs lehetőséget kell átvizsgálni (pl.: szimbolikus deriválás).[2] A mesterséges intelligencia kutatói olyan programokat is készítettek, amelyek egyszerű matematikai alapelemekből (pl. halmazok) komoly felfedezéseket tudtak tenni (pl. *Goldbach-sejtés*). (Hofstadter [1998]).

7. *Kutatás*. Utolsónak maradt az a cél, amelyet a leginkább fontosnak tartunk. A szimulációs módszerek ugyanis a kutatásban, az új összefüggések felfedezésében is igazi segítséget nyújthatnak. Az értekezés második részében elsősorban ennek bemutatására törekszünk.

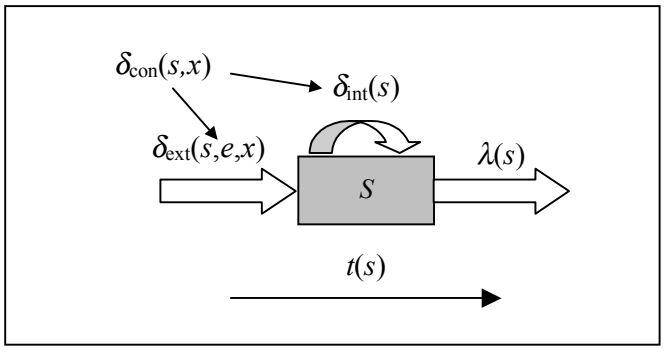
[2] A véges szó sajnos nagyon is meghatározott korlátokat jelent. A *Ramsey-féle R*(5,5) szám megtalálásához 10^{400} esetet kellene megvizsgálni, ami több számításidőt venne igénybe, mint az univerzum életkora, még akkor is, ha a Föld összes számítógépe a nap 24 órájában ezt a problémát számolná! (Ld.: Leader [2001])

A szimulációs módszertan segítségével vizsgált modelleket (röviden: szimulációs modelleket) több szempontból kategorizálhatjuk annak alapján, hogy a háttérben meghúzódó matematikai modell milyen tulajdonságú. Az első kérdés, hogy az *idő* a modellben folytonos, vagy diszkrét változó-e. A legtöbb közgazdasági modellben az idő folytonos változóként szerepel. Nem mindegy azonban, hogy milyen gyakran van lehetőség a szimuláció állapotváltozóit megfigyelni, és beavatkozásokat végezni. Abban az esetben, ha ez csak diszkrét időpillanatokban történhet, – amint az általában egy közgazdasági modellben tapasztalható, – akkor *diszkrét esemény szimulációról* beszélünk. Ha bármely pillanatban lehetőségünk van a beavatkozásra és a megfigyelésre, akkor *folytonos szimulációról* beszélünk. A szimulációban szereplő állapotváltozókat általában folytonosnak tekintjük, de vannak olyan modellek is, amelyben kifejezetten szimbolikus (pl. mesterséges intelligencia) vagy csak diszkrét numerikus (pl. digitális áramkörök) értéket vehetnek fel.

Építhetünk természetesen olyan modelleket is, amelyekben az idő nem szerepel. Ezek a *statikus* modellek. A statikus modellek szimulációja leggyakrabban a statisztikai, ökonometriai alkalmazások esetében játszik fontos szerepet (például a *p-értékek* numerikus előállításánál, vagy egy erőfüggvény alakjának meghatározásánál, Pataki [2001]).

A harmadik legfontosabb kritérium, ami alapján a szimulációs modelleket megkülönböztetjük, hogy az *sztochasztikus* vagy *determinisztikus*. Az utóbbira jó példa a determinisztikus differenciálegyenletek numerikus megoldásánál használt szimuláció.

Mi a továbbiakban olyan sztochasztikus, dinamikus szimulációs modellekkel foglalkozunk, amelyben a megfigyelési és a beavatkozási időpontok diszkrétek, azaz *diszkrét események szimulációjával*. Az állapotváltozók tekintetében nem teszünk megkötéseket. Az ilyen rendszerek szimulációs modelljét formálisan is meg lehet határozni.[3] A formális leírásnak két szintje van: az atomi (*atomic*) szint és az összetett (*coupled*) szint.



2.2. ábra: Az atomi szimulációs modell reprezentációja

[3] Cho, Cho [1997] munkára támaszkodunk.

Az atomi szint a következő:

$$M = \langle X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \delta_{\text{con}}, \lambda, t \rangle, \tag{2.1}$$

ahol: X : a külső események halmaza,
S : a szekvenciális állapotok halmaza,
Y : az outputok halmaza,
$\delta_{\text{int}} : S \to S$: belső átmeneti leképzés,
$\delta_{\text{ext}} : Q \times X \to S$: külső átmeneti leképzés,
$\delta_{\text{con}} : S \times X \to S$: torlódási átmeneti leképzés,
$\lambda : S \to S$: output leképzés,
$t : S \to R$: időzítés függvény, és
$Q = \{(s,e) \mid s \in S, 0 \le e \le t(s)\}$, ahol e az utolsó állapotváltozás óta eltelt idő.

A 2.2. ábrán láthatjuk az atomi szint modelljének felépítését. A formális modell szemléltetésére példaképpen vegyünk egy kiszolgálási feladatot.[4] Tegyük fel, hogy egy csomag beérkezik egy sorbanállási rendszerbe. Ennek következtében a rendszer s állapotváltozója, jelen példánkban a *várakozási sor hossza* 1 egységgel növekszik. Ezt a δ_{ext} függvény valósítja meg. Ezután a t függvény által megadott idő elteltével a modell megvizsgálja a sor hosszát. Néhány csomagot eltávolít, és ennyivel csökkenti az állapotváltozó értékét, amelyet formálisan a δ_{int} függvény valósít meg. A δ_{con} függvény eldönti, hogy mi történjen abban az esetben, ha egyszerre jelentkezik a külső és belső esemény, (például meghatározza δ_{ext} és δ_{int} sorrendjét). A λ függvény végül előállítja a kimeneti változót, jelen esetben például a várakozási sor átlagos hosszát.

Az összetett modell formálisan két alapvető objektumból áll. Komponensekből (amelyek lehetnek atomi szintű vagy összetett modellek) és az ezeket összekapcsoló struktúrából. A struktúra formálisan a következő:

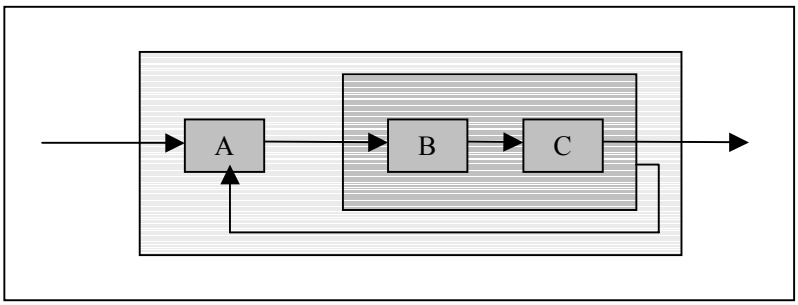
$$DN = \langle D, \{M_i\}, \{I_i\}, \{Z_{ij}\} \rangle, \tag{2.2}$$

ahol D jelöli a komponensek egyedi azonosítóját,
M_i az atomi – vagy magasabb – szintű modelleket,
I_i az egyes komponenseknek az összetett modellre vonatkozó hatását,
Z_{ij} pedig a komponensek egymásra vonatkozó hatását, hierarchiáját.

A 2.3. ábrán egy olyan sémát láthatunk, amely egy sorba kötött kiszolgálási rendszer erőforrásait optimalizálja. Az „A” modul dönt arról, hogy az erőforrások hány százalékát kapja a „B” és hány százalékát a „C” kiszolgálási egység. Az „A” modul

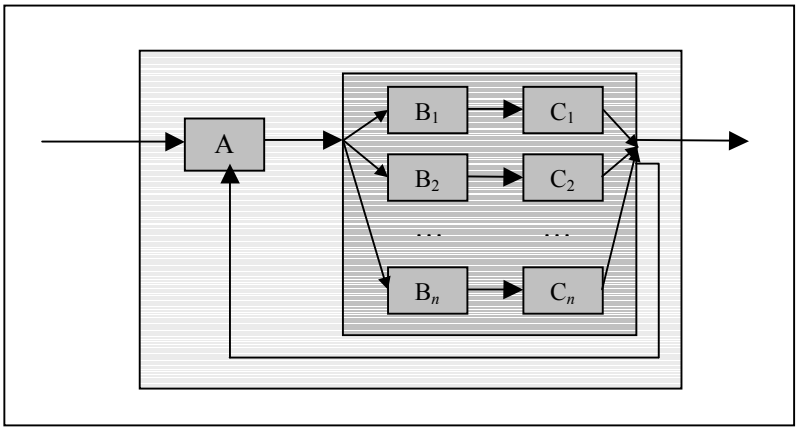
[4] Kiszolgálási folyamatok szimulációjával kapcsolatban ld. például Benedek, Molnár [1996].

visszacsatolásként megkapja a „BC” modul outputját és a belső függvények segítségével eldönti, hogy elérkezett-e az optimumba, vagy további futtatásra van szükség. Ha újabb futtatás kell, akkor új erőforrás allokációt határoz meg.



2.3. ábra: Az összetett modell reprezentációja

A 2.3. ábra jól jellemzi egy determinisztikus rendszer szimulációját, de sztochasztikus modell esetén az egyes kimenetek mindig csak egy lehetséges megvalósulást (*replikációt* vagy *lefutást*) jelentenek. A sztochasztikus modellek kimeneti változója is nyilvánvalóan sztochasztikus, ezért több ismétlésre (replikációra) van ahhoz szükség, hogy ennek a változónak az eloszlását megismerjük. Így a 2.3. ábra úgy módosul, hogy a „BC” rész *n*-szer valósul meg. Az „A” modul feladata lesz az, hogy eldöntse milyen visszacsatolást generál a „BC” kimenetekből, például egyszerű átlagot (*várható érték becslést*). Egy sztochasztikus összetett modell reprezentációja látható a 2.4. ábrán.



2.4. ábra: A sztochasztikus összetett modell reprezentációja

Nézzük, milyen alapvető lépésekből áll a szimulációs modellezés:

- Probléma és rendszer definiálása,
- Modellkoncepció kidolgozása (formális modell),
- Elsődleges kísérletek megtervezése,
- Inputanalízis és input adatok generálása,
- Modell elkészítése (programozás),
- Verifikáció, validáció és modell kalibrálás,
- Kísérletezés,
- Output analízis és interpretálás.

A következő alpontokban részletesen mutatjuk be ezeket a lépéseket. Az ismertetést Benedek [1999] modellje segítségével végezzük, mivel ebben a publikációban minden fontos lépés megtalálható.[5]

2.2. Probléma és rendszer definiálása

Minden szimulációs modellezés első lépése, hogy definiáljuk azt a problémát és a probléma keretrendszerét, amely a kutatás középpontjában áll. Tegyük fel például, hogy azt a feladatot tűzzük ki, hogy egy részvényre vonatkozó vételi opció (*call option*) árát kívánjuk meghatározni.[6] Ebben az esetben a

$$C(S_T,t) = e^{-r(T-t)} \max\{S_T - E, 0\} \tag{2.3}$$

értéket kell meghatározni, ahol

C az opció értékét,
S_t a t időpontbeli részvényárfolyamot,
E a kötési árfolyamot,
r a piaci kamatlábat,
T-t pedig a hátralévő futamidőt jelenti.

A probléma analitikus kezeléséhez igen erős és a valóságban nem teljesülő feltételezésekkel kell élni. Először is a részvényárfolyamtól megköveteljük, hogy *általánosított Brown-mozgást* kövessen, konstans – időben állandó – *drifttel* (μ) és *volatilitással* (σ), azaz

$$\begin{aligned} dS &= \mu S dt + \sigma S dZ \\ dZ &= \varepsilon\sqrt{dt}, \qquad \varepsilon \sim N(0,1) \end{aligned} \tag{2.4}$$

[5] Szeretnénk hangsúlyozni, hogy itt a Benedek [1999] cikkben ismertetett opcióárazás modell csak illusztrációként szerepel, a részletekre és az eredmények bemutatására nem térünk ki.
[6] Igen részletesen tárgyalja ezt a problémát Hull [1993].

Megköveteljük továbbá a piaci kamatláb állandóságát, a részvény tökéletes oszthatóságát, és zéró osztalékfizetését, a piaci kamatlábon történő kölcsönvétel és kölcsönadás lehetőségét, részvényeladást jövőbeli teljesítéssel (*short selling*), a folytonos kereskedési lehetőséget, az adók és a tranzakciós költségek hiányát. Feltételezzük továbbá, hogy *európai típusú* opcióról van szó és a piacon nincs lehetőség *arbitrázsra*. Ilyen megszorító feltételezések mellett meg lehet határozni a C értékét T, t, r, E, σ és μ függvényében. (Black, Scholes [1973]).[7] Az egyes feltételek feloldására számos kutató adott analitikus, illetve numerikus megoldást. (Ezek közül a leglényegesebbek Cox, Ross [1976], Hull, White [1987], Merton [1973], és Cox et al. [1979]). A Benedek [1999] cikkben szimláció segítségével határoztuk meg az opció értékét abban az esetben, ha a részvény árával arányos tranzakciós költséget kell fizetni a kereskedés pillanatában.

A fentiek segítségével definiáltuk a közgazdasági (jelen esetben pénzügyi) problémát, a megoldáshoz szükséges rendszer pedig egy *sztochasztikus diszkrét esemény szimulátor*. A későbbiekben látni fogjuk, hogy a rendszert ki kellett egészíteni egy *numerikus optimalizáló rendszerrel*, amely a genetikus algoritmusra épül.

2.3. Formális modell

A formális modell elkészítése során olyan leírást kell készíteni, amely segítségével bárki képes a meghatározott szimulációs feladat programját elkészíteni. A programozáshoz igen hasznos segítség a *folyamatábra*.

Az előző példát folytatva a szimulációhoz szükséges paraméterek a következők voltak:

E : az opció kötési árfolyama,
r : a kockázatmentes kamatláb,
Q : a tranzakciós költség nagysága százalékban,[8]
S_0 : a részvény kezdeti árfolyama (az opció értékelésének pillanatában),
μ és σ : a részvény driftje és volatilitása és
T : az opció lejáratának időpontja.

---

[7] Ez a *Black-Scholes formula*:

$$C(S,t) = SN(d_1) - Ee^{-r(T-t)}N(d_2)$$

$$d_1 = \frac{\ln(S/E) + (r + \sigma^2/2)(T-t)}{\sigma\sqrt{T-t}} \quad ,$$

$$d_2 = \frac{\ln(S/E) + (r - \sigma^2/2)(T-t)}{\sigma\sqrt{T-t}} = d_1 - \sigma\sqrt{T-t}$$

ahol $N(.)$ a standard normális eloszlás eloszlásfüggvénye.

[8] Azaz $Q = 0.01$ esetén két darab 100 Ft értékű részvény eladása, illetve vásárlása 2-2 Ft extra költséget jelent.

A szimuláció során a három állapotváltozót használtunk:

P_t : a szimulált befektető egyenlege t időpontban, $P_0 = 0$,
S_t : a részvény árfolyama t időpontban,
Δ_t : az opció fedezeti aránya t időpontban.

Szimulációs modellünkben egy képzeletbeli befektető $t = 0$ időpontban elad egy darab vételi opciót és vásárol Δ_0 darab részvényt. A vásárlást kockázatmentes kamatlábon adott hitel segítségével valósítja meg. Ezután minden időpillanatban megfigyeli a részvény aktuális árfolyamát (S_t) és megváltoztatja a portfóliójában szereplő részvények számát Δ_t mennyiségre. Amennyiben ez ismételten vásárlást jelent, akkor további hitelt vesz fel, ha eladást, akkor törleszt. Ezt a technikát *dinamikus fedezési* stratégiának (*dynamic hedging*) nevezik. Az opció lejártakor ($t = T$) helyt kell állnia az opciónál, ami $\max\{S_T - E,\ 0\}$ további kiadást jelent. (Ha ugyanis a részvény ára magasabb a kötési árfolyamnál, akkor az opciót lehívják, így a teljesítésekor a részvényár és a kötési árfolyam különbségét kell kifizetnie az opció eladójának. Ha azonban a részvényárfolyam nem éri el a kötési árfolyamot, akkor az opciót nem fogják lehívni, így az eladó nem szenved veszteséget.) Végül a befektető eladja portfóliójában szereplő Δ_{T-1} részvényt is. A maradék összeg diszkontált értéke adja meg azt az opcióárat (C), amely kifizetése esetén a befektető pontosan zéró nyereséget/veszteséget realizál (*Arbitrázs-mentesség*).

A szimuláció futtatásához tisztázni kell a részvényárfolyamok generálásának képletét, amely a következő:[9]

$$S_{i+1} = S_i \, \mathrm{e}^{\left(\mu - \sigma^2/2\right)(1/Y) + \sigma\varepsilon\sqrt{1/Y}}, \qquad \varepsilon \sim N(0,1). \tag{2.5}$$

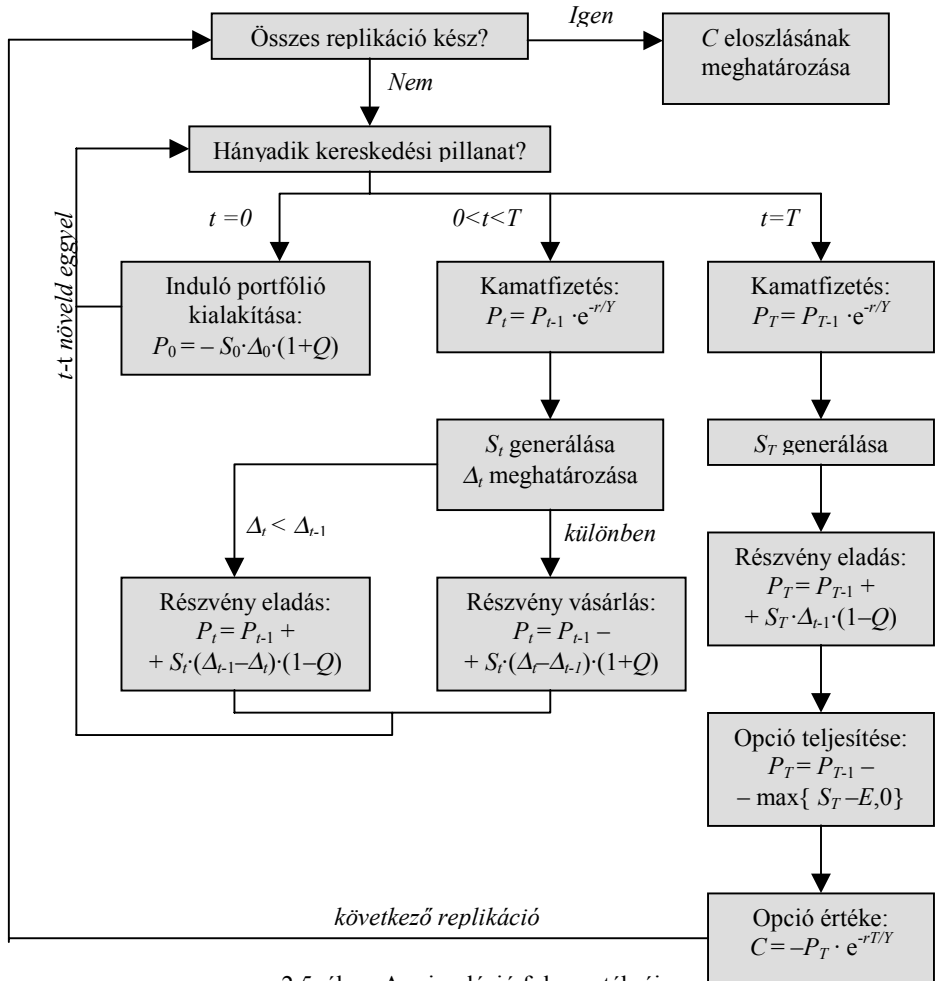
Vegyük észre, hogy az időpont beosztások finomítása az Y paraméter segítségével történik. Mivel a részvény paramétereit éves szinten adtuk meg, ezért Y az jelenti, hogy hány részintervallumra osszuk fel – egyenletesen – az egy évet. (Ha például naponta adunk lehetőséget a portfólió fedezésére, akkor $Y = 360$, ha naponta tízszer, akkor $Y = 3600$. Természetesen T-t is ebben a „mértékegységben" kell mérnünk, azaz ha az opció futamideje egy hónap, akkor az előbbi esetben $T = 30$, míg az utóbbiban $T = 300$.) Nem nehéz belátni, hogy az idő végtelen finomításával a (2.5) formula pontosan a (2.4) sztochasztikus folyamatot eredményezi.[10] A (2.5) formulából jól látható, hogy minden újabb részvényár szimulálásánál egy standard normális eloszlású értéket kell generálnunk. Ennek módszerét a következő alpontban mutatjuk be.

[9] Ennek levezetését ld. Benedek [1999].
[10] Ld. például Benedek [1998].

Végül meg kell határoznunk azt az eljárást, aminek segítségével a dinamikus fedezést megvalósítjuk. Ez a következő:

$$\Delta_t = \frac{\partial C}{\partial S} = \frac{N\left(\ln\left(S_t/E\right) + \left(r + \sigma^2/2\right)\left(T-t\right)\right)}{\sigma\sqrt{T-t}}, \tag{2.6}$$

ahol $N(.)$ a standard normális eloszlás eloszlásfüggvénye. Miután az output változó (C) a sztochasztikus szimuláció miatt valószínűségi változó, ezért nem elegendő egy értéket vizsgálni. A szimulációt független véletlenszámok segítségével többször le kell futtatni. (Ezeket nevezi a szakirodalom *replikációknak*, vagy *mintának*. A replikációk számát *mintaméretnek* is hívjuk). Készítsük el a szimuláció folyamatábráját!



2.5. ábra: A szimuláció folyamatábrája

2.4. Elsődleges kísérletek megtervezése

A szimuláció futtatásához természetesen meg kell határozni bizonyos paramétereket. Az elsődleges kísérleteket érdemes úgy megtervezni, hogy a kimenetelét könnyű legyen valamilyen már meglévő értékhez viszonyítani. Ilyen érték lehet a szakirodalomban szereplő számítási eredmény, illetve bizonyos esetekben az analitikus eredmény. Ilyenkor olyan egyszerűsítéseket feltételezünk, amelyet a későbbi kísérletek során fel lehet oldani, azonban az ellenőrzéshez és a hibakereséshez jól lehet alkalmazni.

Az opcióár szimulációban a következő feltételezésekkel éltünk az elsődleges kísérletek megtervezése során:

$E = 100$	$r = 0.05$	$S_0 = 100$
$\mu = 0.12$	$\sigma = 0.3$	$T/Y = 30$ nap

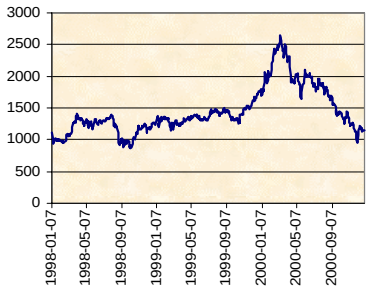
A mintaméretet 5000 replikációnak választottuk és a szimuláció kimeneti változójának a kapott opcióárak átlagát és szórását választottuk. A tranzakciós költség (Q) rátáját először zérusra, majd 1%-ra állítottuk. A zérus tranzakciós költség szimulációjának vissza kell adnia a *Black-Scholes-formula* értékét, hiszen ebben az esetben analitikus eredmények állnak a rendelkezésünkre.

2.5. Inputanalízis és input adatok generálása

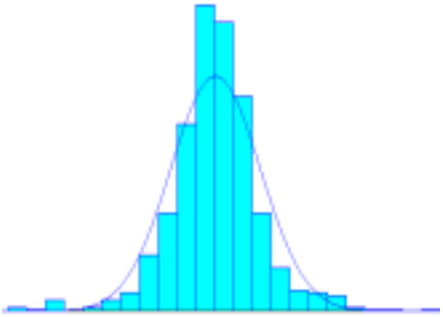
A valóságos rendszer működésének pontos számítógépen belüli előállításához a valóságban lezajló folyamatokat részleteiben kell megfigyelni. Fel kell tárni az összefüggéseket és jellemezni kell a véletlen jelenségeket. Ez a feladat az inputanalízis, vagyis azon adatok elemzése, amelyek a rendszerbe beérkeznek. Az elemzéshez elsősorban statisztikai módszereket alkalmaznak, különösen nagy figyelmet szentelve az eloszlások becslésének. Az utóbbi időben elterjedt másik vizsgálati módszer az *adatbányászat*, amelyet akkor érdemes alkalmazni, ha a megfigyelési adathalmaz rendkívül nagy, és valószínűsíthető, hogy viszonylag bonyolult, nemlineáris összefüggések húzódnak meg bennük, amelyeket célszerű felhasználni a szimulációban.[11] Az adatbányászati elemzések bemutatása sajnos meghaladja az értekezés kereteit, érdemes azonban annyit megjegyezni róla, hogy a módszertan szintén a mesterséges intelligencia algoritmusait használja; olyan algoritmusokat, mint amelyeket a 3. fejezetben mutatunk be.

[11] Az adatbányászat alkalmazásáról számos könyv áll rendelkezésre. Pl.: Bigus [1996], Berry, Linoff [2000]. Magyar nyelven ld. Benedek [1999b] bevezető jellegű cikkét.

Az opcióár szimulációhoz szükséges inputanalízis elsősorban a részvényárfolyam eloszlásának vizsgálatát jelenti. A Benedek [1999] publikációban ilyen típusú vizsgálatot nem végeztünk, mivel a szakirodalomban alkalmazott normalitás feltételt elfogadtuk, az alkalmazott μ és σ paraméter értékeket a szakirodalomból vettük (Hull [1993]). Most azonban bemutatjuk, hogy hogyan kellene eljárnunk ezek ismerete hiányában. Vegyük például a *Matáv* részvényt a *Budapesti Értéktőzsdéről*. A 2.6. ábrán ábrázoljuk a részvényárfolyamokat (S_t) 1998. január 7.-től 2000. december 27.-ig. A 2.7. ábrán pedig megvizsgáljuk a részvényárfolyam növekmények ($\ln(S_t / S_{t-1})$) eloszlását.



2.6. ábra: A Matáv árfolyama



2.7. ábra: Az árfolyamnövekmények eloszlása

Első feladat az, hogy megvizsgáljuk, vajon tényleg normális eloszlást követnek-e az árfolyamnövekmények. Az adatokra egy 0.00004725 várható értékű és 0.011812 szórású *Gauss-görbét* illesztettünk, így a teljes négyzetes hiba 0.008150 volt. Sajnos azonban minden teszt (χ^2 teszt, *Kolgomorov-Szmirnov* teszt, *Jarque-Bera* teszt) elutasítja az adatsor normalitását. Ennek ellenére ebben a példában fogadjuk el a normális eloszlás feltételezését, mivel a szimulációs problémát nem a *Matáv* részvényre vonatkozó opció kiszámítása céljából vizsgáljuk, hanem elméleti szempontból a tranzakciós költség vizsgálata érdekében.[12] Természetesen további fontos teszteknek kellene még alávetni az adatokat (pl.: függetlenség, homoszkedaszticitás), amelyektől eltekintünk, de még egyszer felhívjuk a figyelmet ezek fontosságára, hiszen hibás feltevések mellett hibás következtetésekre jutunk a szimuláció során.[13]

Az illesztett eloszlás és a (2.5) formula segítségével megállapíthatjuk μ és σ paramétereket, amelyek rendre 0.094 és 0.406.

[12] Amennyiben ezt a részvényt kívánjuk felhasználni, úgy további eloszlásokat kell megpróbálni illeszteni, esetleg meg lehet tartani az empirikus eloszlás feltételezését is. Ebben az esetben magából a hisztogramból kell mintavételezni. A *Matáv* adatsorra végül alacsony szabadságfokú *t-eloszlás* illesztése bizonyult hatékonynak (Benedek et. al [2001]).

[13] Statisztikai vizsgálatokhoz ld. pl.: Hunyadi, Vita [1991], Móry, Székely [1986].

Az inputanalízis elvégzése után következő feladat az, hogy hogyan lehet ezeket a véletlen sorozatokat újra előállítani, azaz hogyan történik a *véletlenszám generálás*. Ez a terület a szimulációs modellezés egyik leginkább kutatott témaköre, és gyakorlatilag minden szimulációval foglalkozó tankönyv részletesen foglalkozik vele. A továbbiakban a véletlenszámok ismertetésénél Kátai [1981] és Knuth [1987] munkáira hivatkozunk. (A módszertani eredmények mellett kiváló történeti és alkalmazási áttekintés található Knuth [1987] második kötetében, a *Szeminumerikus algoritmusokban*.)

Véletlenszám-generálásra két módszer áll rendelkezésre. Az egyik módszer az analóg generátor, amikor valamilyen periféria véletlen viselkedése határozza meg a sorban következő számot. Ilyen lehet például egy számítógépbe épített inga, amely állandó kitérései véletlen sorozatokat hozhatnak létre. Az ilyen generátorokkal a legnagyobb probléma az, hogy lehetetlen rekonstruálni egy korábbi folyamatot. Sok esetben ugyanis szükséges lehet arra, hogy ugyanazokkal a véletlenszámokkal megismételjünk egy-egy kísérletet. A másik módszer számelméleti alapokon nyugszik. Matematikai módszer segítségével egyenletes eloszlású változókat generálunk, majd ezután ezek transzformálása segítségével juthatunk a legkülönbözőbb eloszlásokhoz.

Nézzük például Neumann János által javasolt „négyzetközép-módszert”. A véletlen sorozat úgy épül fel, hogy az utolsó számot négyzetre kell emelni, és ennek a középső jegyei alkotják a következő számot. Például négyjegyű számok generálása esetén:

$V_0 = 5784 \quad \rightarrow \quad V_0^2 = 33\mathbf{4546}56$
$V_1 = 4546 \quad \rightarrow \quad V_1^2 = 20\mathbf{6661}16$
$V_2 = 6661 \quad \rightarrow \quad V_2^2 = 44\mathbf{3689}21$
$\ldots \qquad\qquad\quad \ldots$

Felmerül azonban a kérdés, hogy egy algoritmus által előre meghatározott képlet szerint kiszámított számok lehetnek-e véletlenszerűek. Valójában nem azok, viszont úgy viselkednek, mintha tényleg egyenletes eloszlásúak lennének. (Ezért ezeket a sorozatokat a szakirodalom *pszeudovéletlennek* vagy *kvázivéletlennek* nevezi.) A pszeudovéletlen sorozatokkal két gond lehet. Az egyik, hogy ciklusba kerül, azaz egy meghatározott sorozat után újból ugyanazokat a számokat generálja ugyanabban a sorrendben. (Például a négyzetközép-módszer esetén induljunk a 3792-ből!) Belátható, hogy a négyzetközép-módszer tetszőleges indulóérték esetén hamar ciklusba torkollik. A másik probléma, hogy a módszer nem ad olyan számsorozatot, amely valóban független egyenletes eloszlású. A véletlenszám-generáló eljárásokat tehát ebből a két aspektusból kell megvizsgálni.

Napjainkban a leggyakrabban használt egyenletes eloszlást adó numerikus véletlenszám-generátor a *lineáris kongruencia módszer*, amelyet Lehmer [1951] vezetett be. Vegyük a következő sorozatot:

$$V_{i+1} = (aV_i + c) \bmod m, \tag{2.7}$$

ahol minden paraméter nem-negatív egész, mégpedig:

m : a modulus (maradékos osztó):	$m > 0$,
a : az együttható:	$m > a \geq 0$,
c : a növekmény:	$m > c \geq 0$,
V_0 : a kezdőérték (*seed*):	$m > V_0 \geq 0$,

Ezt nevezzük lineáris kongruencia-sorozatnak. Az előállított szám mindig az m-mel történő osztás maradéka. Például, ha $m = 10$ és $V_0 = a = c = 7$, akkor a sorozat a következőképpen fest:

7, 6, 9, 0, 7, 6, 9, 0, …

Jól látható, hogy az ilyen típusú megválasztás esetén a sorozat egy négy elemből álló ciklust hoz létre. A lineáris kongruencia-sorozatban kialakuló ciklus elemszámát *periódushossznak* nevezzük. A következőkben kimondunk néhány fontos tételt:

2.1. Tétel: *Minden $V_{i+1} = f(V_i)$ eljárással megadott korlátos sorozatnál előbb vagy utóbb ciklus jelentkezik.*

Így nyilvánvaló, hogy a (2.7) esetében is biztosan fellép az ismétlődés. Kérdés azonban az, hogy lehet-e kellően nagyra választani a periódushosszt annak érdekében, hogy ez ne okozzon gondot a véletlen sorozat generálásánál. Könnyű belátni a következő tételt:

2.2. Tétel: *A* (2.7) *lineáris kongruencia-sorozat maximális periódushossza pontosan m.*

Ha egy ilyen sorozat periódushossza m, akkor *teljes periódusú* (*full period*) *lineáris kongruencia-sorozatnak* nevezzük.

Ezek szerint számunkra olyan paraméterek szükségesek, ahol m értéke nagy. Sok algoritmus esetében ezért m-et olyan nagynak szokták választani, mint amekkora a számítógép processzorának számábrázolási kapacitása. (32 bites processzor esetén például $m = 2^{32}$). A nagy m mellett olyan egyéb paraméterválasztás a célszerű, hogy a sorozat teljes periódusú legyen. Így kihasználjuk a maximális periódus lehetőséget, továbbá biztosítjuk, hogy minden egyes szám pontosan egyszer fog előfordulni 0 és m–1 között egy cikluson belül. A paraméterek helyes megválasztásához a következő – nehéz számelméleti – tételt kell felhasználni.

2.3. Tétel: *A (2.7) lineáris kongruenciasorozat periódushossza pontosan akkor m, ha:*

1. *c* relatív prím *m*-hez,
2. $b = a - 1$ többszöröse *p*-nek *m* minden *p* prímosztójára,
3. *b* többszöröse 4-nek, ha *m* többszöröse 4-nek.

A tételek bizonyítása megtalálható Knuth [1987] munkájában (II / 24 – 35). Bizonyítás helyett nézzünk egy példát, ahol $m = 8$, ezért $p = 2$, tehát *b* legyen 4, így $a = 5$, $c = 3$ és induljunk az 1-ből. Ekkor a következő sorozatot kapjuk:

1, 0, 3, 2, 5, 4, 7, 6, 1, 0, 3, 2, 5, 4, 7, 6, …

A sorozat tényleg teljes periódusú, a független egyenletes eloszlás próbáját azonban aligha állná ki. Ezért rendkívül fontos az, hogy statisztikai próbákkal megvizsgáljuk a generált sorozatot. A leggyakrabban a következő teszteket szokták elvégezni:

- χ^2-próba,
- Kolgomorov–Szmirnov-próba,
- Gyakoriság-próba,
- Sorozat-próba,
- Hézag-próba,
- Partíció-próba (pókerpróba),
- Sorozatkorreláció-próba,
- Spektrálpróba.

A legtöbb teszt ismertetése megtalálható általános statisztikai könyvekben, azonban a korábban többször is idézett Knuth [1987] könyvben ezeket megvalósító, erőforrás-takarékos algoritmusok is szerepelnek. A ma leginkább megbízható algoritmusok megtalálhatók Press et al. [1992] művében, ahonnan a 10^8 periódushosszú *ran1* és a 10^{16} periódushosszú *ran2* algoritmusokat alkalmaztuk a pénzügyi szimuláció esetében. Bizonyos szimulációk esetén elképzelhető, hogy még a *ran1* vagy a *ran2* algoritmusoknál is hosszabb periódusú – vagy gyorsabb – eljárásra van szükség. A mai legfrissebb kutatások számos ilyen algoritmust ajánlanak, amelyek általában már nem kizárólag a lineáris kongruencia módszerre épülnek. Eddigi közgazdasági szimulációink során azonban ezekre a módszerkre még nem volt szükségünk, így bemutatásuktól eltekintünk.

Sikerült tehát független egyenletes eloszlású számsorozatokat generálni számelméleti módszerek alkalmazásával. A szimulációs alkalmazásokhoz azonban különböző eloszlású változókra lehet szükség. Ezeket az eloszlásokat egyenletes eloszlású változókból transzformálva készíthetjük el, leggyakrabban a következő módszereket

alkalmazva: *direkt módszer*, *kizárás módszere* (*acceptance-rejection method*), *konvolúciós módszer* és speciális módszerek.

Direkt módszer. Tetszőleges θ folytonos valószínűségi változó esetén jelölje F_θ θ eloszlásfüggvényét, f_θ pedig θ sűrűségfüggvényét. A következő tételek viszonylag egyszerűek, bizonyításuk a legtöbb valószínűségszámítással foglalkozó tankönyvben megtalálható. (Rényi [1973])

2.4. Tétel: *Legyen ξ folytonos valószínűségi változó, g pedig egy folytonos, valós, monoton növekvő függvény. Legyen továbbá $\eta = g(\xi)$. Ekkor:*

$$F_\eta(x) = F_\xi\left(g^{-1}(x)\right). \tag{2.8}$$

Ebből következik az alábbi tétel:

2.5. Tétel: *Legyen ξ folytonos valószínűségi változó egyenletes eloszlású a* [0,1] *intervallumon, $F(x)$ pedig egy tetszőleges folytonos eloszlásfüggvény. Ekkor $\eta = F^{-1}(\xi)$ olyan valószínűségi változó, amelynek eloszlásfüggvénye F.*

A tétel alapján már látjuk miért volt fontos az egyenletes eloszlású valószínűségi változó generálása. Ennek segítségével bármilyen más eloszlást készíthetünk, feltéve, hogy ismerjük az eloszlás eloszlásfüggvényének inverzét. Nézzünk egy példát, az *exponenciális eloszlást,*[14] amelynek az eloszlásfüggvénye a következő:

$$F(x) = \begin{cases} 1 - \mathrm{e}^{-x/b}, & \text{ha } x \geq 0 \\ 0, & \text{különben} \end{cases} \tag{2.9}$$

Ezért könnyű előállítani F inverzét: $x = F^{-1}(u) = -b\ln(1-u)$. Sajnos a legtöbb esetben nem ilyen egyszerű a helyzet, például a számunkra igen fontos normális eloszlás esetén nem adható meg – analitikus formában – az eloszlásfüggvény.[15]

A folytonos valószínűségi változó esetét megvizsgáltuk, nézzük milyen lehetőségek vannak a diszkrét véletlen változók előállítására!

[14] Az exponenciális eloszlás a sorbanállási és készletgazdálkodási modellek tekintélyes részénél szerepel, ezért igen fontos eloszlás.
[15] Több program is közelítő függvény segítségével, direkt módszerrel generál normális eloszlást. Ezek megbízhatósága – különös tekintettel az eloszlás széleire – igen különböző.

2.6. Tétel: *Legyen ξ diszkrét valószínűségi változó, mégpedig:*

$$p_m = P(\xi = x_m), \qquad m = 0, 1, 2, \ldots \tag{2.10}$$

Legyen továbbá:

$$S_m = \sum_{k=0}^{m} p_k, \quad I_k = [S_{k-1}, S_k), \quad k \geq 0, \quad S_{-1} = 0. \tag{2.11}$$

Ezért:

$$p_m = P(S_{m-1} \leq u < S_m) = P(u \in I_m). \tag{2.12}$$

Ekkor a következő eljárással egy u egyenletes eloszlású változó segítségével pontosan a ξ diszkrét eloszlású valószínűségi változót generáljuk:

$$\xi = \begin{cases} x_0, & \text{ha } u \in I_0 \\ x_1, & \text{ha } u \in I_1 \\ & \ldots \end{cases} \tag{2.13}$$

Például egy dobókocka dobásait rendkívül egyszerűen szimulálhatjuk, ha az egyenletes eloszlású $u < {}^1/_6$, akkor a kockadobás 1-es, ha ${}^1/_6 \leq u < {}^2/_6$, akkor 2-es stb. A direkt módszer utolsó tétele megmutatja, hogy tetszőleges eloszlás generálható egyenletes eloszlás segítségével:

2.7. Tétel: *Legyen ξ_1 eloszlásfüggvénye F_1, ξ_2-é pedig F_2. Tegyük fel továbbá, hogy:*

$$F(x) = pF_1(x) + qF_2(x), \quad p + q = 1, \quad p, q \geq 0. \tag{2.14}$$

Tegyük fel, hogy ξ lehetséges értékei 1 *és* 2*, rendre p és q valószínűséggel, továbbá ξ független ξ_1 és ξ_2 változóktól. Definiáljuk η-t:*

$$\eta = \begin{cases} \xi_1, & \text{ha } \xi = 1 \\ \xi_2, & \text{ha } \xi = 2 \end{cases} \tag{2.15}$$

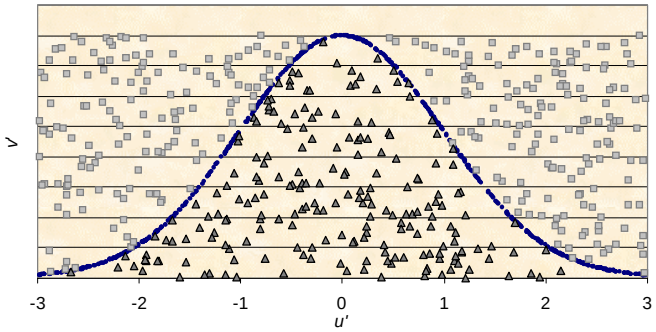
Ekkor η eloszlásfüggvénye F, továbbá mivel minden monoton növekvő függvény felbontható két monoton növekvő függvény összegére úgy, hogy az egyik folytonos, a másik pedig diszkrét (tiszta ugró), ezért minden eloszlásfüggvény felírható (2.14) *alakban, ahol F_1 folytonos, F_2 pedig diszkrét eloszlásfüggvény.*

Kizárás módszere. A módszert nevezik még *Monte-Carlo módszernek* is, bár ezt a nevet általában a hasonló elven működő numerikus integrálásra szokták használni. Az eljárás lényege a következő.

Legyen adott *u, v* egyenletes eloszlású független valószínűségi változó a [0,1] intervallumon. Szeretnénk elkészíteni ξ véletlen változót, ahol $\xi \in [a, b]$, sűrűségfüggvénye *g,* és *g* maximumértéke pedig $c \leq 1$. Transzformáljuk először *u*-t *u'*-be, *v*-t *v'*-be, ahol ezek rendre egyenletes eloszlásúak az [*a*, *b*] és a [0, *c*] intervallumon. Ezután vizsgáljuk meg, hogy igaz-e a $v' \leq g(u')$ egyenlőtlenség. (Grafikusan elképzelve vajon a generált (*u' v'*) pont a sűrűségfüggvény alatt vagy felett helyezkedik el.) Ha igaz (alatta), akkor fogadjuk el $\xi = u'$-t, ha nem, akkor generáljunk újabb *u, v* párost.

2.8. Tétel: *Az így kapott ξ valószínűségi változó pontosan olyan eloszlást követ, amelynek sűrűségfüggvénye g, továbbá minden egyes ξ változó előállításához* $2\cdot(b-a)\cdot c$ *számú független egyenletes eloszlású változóra van szükség.*

Ezzel a módszerrel két probléma van. A megvalósításhoz egyrészt ismerni kell *g* sűrűségfüggvényt, másrészt csak korlátos valószínűségi változót tudunk generálni. Ha például standard normális eloszlású valószínűségi változót szeretnénk készíteni, akkor meg kell határozni azt a tartományt, amelynél kisebb, illetve nagyobb számokat 0 valószínűséggel generálunk. Mivel a sűrűségfüggvény szélei nagyon gyorsan konvergálnak nullához, ezért már a [-3, 3] választással elég kicsi hibát vétünk.[16] (3σ–szabály) A 2.8. ábra segítségével könnyen megérthető a 2.8. tétel. Az ábrán háromszöggel jelöltük az elfogadott, négyzettel az elutasított szám-párokat. Az elfogadott szám-párok első koordinátái adják a normális eloszlású valószínűségi változókat.



2.8. ábra: Normális eloszlás generálása a kizárás módszerrel

[16] Természetesen minél több független véletlen változóra van szükségünk, annál nagyobb az így elkövetett hiba valószínűsége. A pénzügyi szimuláció során a [-12, 12] intervallummal próbálkoztunk.

Konvolúciós módszer. Sok eloszlást tudunk más eloszlások összegeként generálni. Így készíthetünk például *Erlangen-eloszlást* azonos várhatóértékű exponenciális eloszlásokat összegezve. Az egyik legegyszerűbb a központi határeloszlás segítségével a normális eloszlás készítése. Ezt mutatjuk most be:

2.9. Tétel: *Adott egy* $y_i \sim U(0, 1)$, *azaz egy egyenletes eloszlású minta a* [0,1] *intervallumon. Transzformáljuk* $y'_i \sim U(-\alpha,\alpha)$*-ra, azaz* $y'_i = 2\alpha y_i - \alpha$. *Ezután n-esével összeadjuk és standardizáljuk őket, azaz:*

$$z = \frac{(y'_1 + y'_2 + \ldots + y'_n)/n}{\sigma/\sqrt{n}}, \quad ahol \quad \sigma^2 = \frac{\alpha^2}{3}. \tag{2.17}$$

Természetesen itt z is indexelt, azaz minta, pontosan annyi elemű, ahány n-es csoportot létre lehet hozni a független egyenletes eloszlású mintából. Ekkor (i) y'_i *független, egyenletes eloszlású a* $[-\alpha,\alpha]$ *intervallumon, várható értéke* 0 *és szórása* σ, (ii) *z normális eloszlást követ, ha n tart a végtelenhez, továbbá* (iii) *z már egészen kicsi n értékekre is igen jól közelíti a standard normális eloszlást, különös tekintettel az eloszlás közepére.*

A tétel (i) része triviális, a (ii) pedig a központi határeloszlás tételének következménye. A (iii) azonban nem triviális és nem található meg a szakirodalomban, ezért ezt a bizonyítást részletezzük:

Bizonyítás: *A bizonyítás során azt mutatjuk meg, hogy z karakterisztikus függvénye gyorsan konvergál a standard normális eloszlás karakterisztikus függvényéhez. Először állítsuk elő* y'_i *karakterisztikus függvényét:*

$$\begin{aligned} k_{y'}(t) &= \mathrm{E}\left(\mathrm{e}^{\mathrm{i}tx}\right) = \int_{-\alpha}^{\alpha} \mathrm{e}^{\mathrm{i}tx} \frac{1}{2\alpha} dx = \frac{1}{2\alpha}\left[\frac{1}{\mathrm{i}t}\mathrm{e}^{\mathrm{i}tx}\right]_{-\alpha}^{\alpha} = \frac{1}{2\alpha\mathrm{i}t}\left(\mathrm{e}^{\mathrm{i}t\alpha} - \mathrm{e}^{-\mathrm{i}t\alpha}\right) = \\ &= \frac{1}{2\alpha\mathrm{i}t}\left(\cos(t\alpha) + \mathrm{i}\sin(t\alpha) - \cos(-t\alpha) - \mathrm{i}\sin(-t\alpha)\right) = \frac{\sin(t\alpha)}{t\alpha} \end{aligned} \tag{2.18}$$

Ekkor a karakterisztikus függvények tulajdonságai miatt könnyen kiszámíthatjuk z karakterisztikus függvényét.

$$z = \frac{\sqrt{n}}{n\sigma}\sum_{i=1}^{n} y_i = \frac{\sqrt{3}}{\alpha\sqrt{n}}\sum_{i=1}^{n} y_i, \tag{2.19}$$

ezért

$$k_z(t) = \left(\frac{\sin\left(t\alpha \frac{\sqrt{3}}{\alpha\sqrt{n}} \right)}{\frac{t\alpha\sqrt{3}}{\alpha\sqrt{n}}} \right)^n = \left(\frac{\sin \frac{t\sqrt{3}}{\sqrt{n}}}{\frac{t\sqrt{3}}{\sqrt{n}}} \right)^n$$

Tudjuk azt is, hogy a sinx Taylor-polinomjának hibatagja rendkívül kicsi a 0 körül, ezért alkalmazzuk ezt a közelítést:

$$\sin x \approx x - \frac{x^3}{6} \tag{2.20}$$

Ekkor:

$$k_z(t) = \left(\frac{\frac{t\sqrt{3}}{\sqrt{n}} - \left(\frac{t\sqrt{3}}{\sqrt{n}} \right)^3 \Big/ 6}{\frac{t\sqrt{3}}{\sqrt{n}}} \right)^n = \left(1 - \frac{\left(\frac{t\sqrt{3}}{\sqrt{n}} \right)^2}{6} \right)^n = \tag{2.21}$$

$$= \left(1 - \frac{3t^2}{6n} \right)^n = \left(1 + \frac{-t^2/2}{n} \right)^n \approx \mathrm{e}^{-\frac{t^2}{2}}$$

■

Ezt a módszert sok generátor alkalmazza normális eloszlású véletlen változó generálásához. A szakirodalom az $n = 12$ esetet már megbízhatónak tartja. A probléma azonban az, hogy ebben az esetben is 12 darab egyenletes eloszlású szám kell, minden egyes normális eloszlású számhoz, ezért az algoritmusnak nagy az időigénye és hamarabb eljutunk a periódushossz végéig. Ezért alkalmaznak egyéb speciális módszereket.

Speciális módszerek. Számos speciális eljárást találunk különböző eloszlások generálására[17] (pl.: *Béta-eloszlás*, *Gamma-eloszlás*, *binomiális eloszlás*, *Poisson-eloszlás*, stb.) Most a normális eloszlásra mutatunk egy lehetséges eljárást, mivel pénzügyi szimulációnkban ezt implementáltuk és a szakirodalom is ezt a módszert javasolja legtöbbször:

[17] Az algoritmusok és bizonyítások tekintetében ismét Knuth [1987] könyvét javasoljuk.

2.10. Tétel: ***Polármódszer.*** *Generáljunk két független egyenletes eloszlású valószínűségi változót, u_1-et és u_2-t. Transzformáljuk őket a* [-1, 1] *intervallumba, majd az így kapott* (v_1, v_2) *változókra vizsgáljuk meg, hogy beleesnek-e az origó körüli egységnyi sugarú körbe. Ha nem, azaz* $S = v_1^2 + v_2^2 \geq 1$, *akkor újra kell kezdenünk az eljárást. Ellenkező esetben legyen:*

$$x_1 = v_1\sqrt{\frac{-2\ln S}{S}}\,, \quad x_2 = v_2\sqrt{\frac{-2\ln S}{S}} \tag{2.22}$$

Az így kapott x_1 és x_2 független standard normális eloszlású véletlen szám.

Bizonyítás: *Először is áttérünk* (v_1, v_2)*-ről azok polárkoordinátáira. (A sugár mindig* 1*-nél kisebb lesz, hiszen csak ezeket fogadtuk el). Ekkor* $v_1 = R\cos\theta$ *és* $v_2 = R\sin\theta$, *továbbá* $x_1 = \sqrt{-2\ln S}\cos\theta$, $x_2 = \sqrt{-2\ln S}\sin\theta$. *Nyilván x_1 és x_2 esetében is áttérhetünk ezek polárkordinátáira, azaz* $x_1 = Q\cos\phi$ *és* $x_2 = Q\sin\phi$. *Ezért igaz, hogy* $\theta = \phi$ és $Q = \sqrt{-2\ln S}$. *Q és* ϕ *független, S egyenletes eloszlású* [0, 1)*-en,* θ *pedig* [0, 2π)*-n. Ebből következik, hogy:*[18]

$$\begin{aligned}
&\Pr(Q < q) = \Pr\left(-2\ln S < q^2\right) = \Pr\left(S > e^{-q^2/2}\right)\\
&\Pr(Q < q) = 1 - e^{-q^2/2}\\
&\Pr(q \leq Q < q + dq) = \frac{d\left(1 - e^{-q^2/2}\right)}{dq} = qe^{-q^2/2}
\end{aligned} \tag{2.23}$$

$$\begin{aligned}
&\Pr(\phi < \vartheta) = \vartheta/2\pi\\
&\Pr(\vartheta \leq \phi < \vartheta + d\vartheta) = 1/2\pi\,.
\end{aligned}$$

Mi a valószínűsége annak, hogy $x_1 < y_1$ *és* $x_2 < y_2$ *?*

$$\begin{aligned}
&\int\limits_{\{(q,\vartheta)\,|\,q\cos\vartheta < y_1,\, q\sin\vartheta < y_2\}} \frac{q}{2\pi}\mathrm{e}^{-q/2}\,dq\,d\vartheta = \frac{1}{2\pi}\int\limits_{\{(x_1,x_2)\,|\,x_1 < y_1,\, x_2 < y_2\}} \mathrm{e}^{-\left(x_1^2 + x_2^2\right)/2}\,dx_1\,dx_2 =\\
&= \left(\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{y_1}\mathrm{e}^{-x_1^2/2}\,dx_1\right)\left(\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{y_2}\mathrm{e}^{-x_2^2/2}\,dx_2\right)
\end{aligned} \tag{2.24}$$

■

[18] A Pr($q < Q$) jelölés annak a valószínűségét jelenti, hogy $q < Q$.

Látható az is, hogy az eljárás az egységsugarú körön kívüli elemeket dobja el, így a területarányok miatt egy darab normális eloszlású szám generálása átlagosan 1.27 számú egyenletes eloszlású változót igényel. Ez az algoritmus a szakirodalomban *gasdev* néven szerepel. (Press et al. [1992]).

Eddig mindig csak egydimenziós eloszlásokról beszéltünk, de szükségünk lehet valószínűségi vektorokra is, ahol az egyes elemek nem függetlenek. Az ilyen típusú problémákra általában választ kapunk a statisztikai szakirodalomból. Példaként nézzük a többdimenziós normális eloszlást, mivel pénzügyi szimulációk során erre gyakran szükség lehet.

2.11. Tétel: *Tegyük fel, hogy egy* **m** *várható értékű* **C** *variancia-kovariancia mátrixszal rendelkező n dimenziós normális eloszlást szeretnénk generálni. Első lépésként generáljunk n darab független standard-normális eloszlású változót* (**x**). *Ekkor az* **y = m + Tx** *transzformálással pontosan a kívánt eloszlást állítottuk elő, feltéve hogy* **C = TT'**.

Az alfejezet utolsó témája a *véletlen keverés*. Ilyen például a kártyalapok megkeverése. Közgazdasági szempontból azért nagyon fontos, mert a keresési modellek (*search models*) nagy része azt feltételezi, hogy az egyes szereplők véletlenszerűen találkoznak egymással, és minden periódusban mindenki találkozik mindenkivel. Ehhez a szereplők véletlen keverését kell előállítani. Hasonló a probléma akkor is, ha egy véletlen sorrendet kívánunk meghatározni. Vegyük a következő esetet: az 1, 2, …, 100 számokat kell véletlenszerűen megkeverni úgy, hogy egy egyenletes eloszlás generátor áll rendelkezésünkre.

Az egyik lehetőség az, hogy a (2.6) tétel segítségével az egyenletes eloszlásból az 1, 2, …, 100 számokat egyforma eséllyel generáló diszkrét eloszlást készítünk. Az első így kapott számot tesszük a sor elejére. Ezután újabb számot generálunk, s ha ez nem egyezik meg az előzővel, akkor a sor második tagjává tesszük, ellenkező esetben újabbat generálunk. A harmadik szám generálásakor már az első kettő számmal kell egyeztetnünk, a negyediknél az első hárommal, stb. Ezzel az algoritmussal véletlen sorrendbe lehet rakni a számokat. A gond az, hogy meglehetősen számításigényes és pazarló ez az algoritmus, hiszen az utolsó esetben már átlagosan száz darab véletlen számot kell generálni ahhoz, hogy a megfelelő utolsót kihúzzuk.

A másik algoritmus, amelyet a későbbiek során is gyakran használunk, ennél jóval gazdaságosabb. A módszert úgy képzelhetjük el, hogy egy „dobozból" átpakoljuk a számokat egy másik dobozba. Az elsőben növekvő sorrendben vannak a számok 1-től 100-ig. Kiválasztunk egy véletlen számot (1 és 100 között), és az annyiadik elemet áttesszük a második dobozba. Az első dobozban így egy hely üresen maradt, ide betesszük ugyanennek a doboznak a századik elemét. A maradék 99-ből megint

kiválasztunk egyet, (természetesen ekkor már csak 1 és 99 közötti véletlen számot kell generálnunk) és áttesszük, stb. Ez az algoritmus nagyságrendekkel gyorsabb, mint az előző.[19]

Ezzel befejeztük az inputanalízissel és véletlenszámokkal foglalkozó alfejezetünket. Még egyszer szeretnénk felhívni a figyelmet arra, hogy az inputanalízis és a véletlen adatok helyes generálása kritikus része a helyes szimulációs modellezésnek.

2.6. Modell elkészítése (programozás)

A programozás témakör rengeteg problémát vet fel. Milyen programozási nyelvet vagy szimulációs környezetet érdemes használni? Milyen algoritmusokat és milyen programkönyvtárakat lehet alkalmazni? Hogyan kell felépíteni egy szimulációs programot? Ezek a kérdések rendkívül sokfelé ágaznak, és megválaszolásukra a jelen értekezésben nincs lehetőség, de számos szimulációs kézikönyv és szimulációs szoftver leírás ad erről részletes leírást, ahol a programozási eljárásokat, részletes algoritmusokat is ismertetik. A leggyakoribb szimulációs környezetek a *C* és a *Pascal* nyelvek (illetve ezek objektumorientált verziói, a *C++* és a *Delphi*), továbbá a felhasználóbarát szimulációs szoftverek, mint az *Arena*, a *GPSS* vagy a *Taylor Enterprise Dynamics*, illetve a matematikai programcsomagok, mint a *MATLAB*, a *MATHEMATICA* vagy a *MAPLE*. Adott esetben még az *Excel* is funkcionálhat szimulációs környezetként – bár ettől már egy közepes szimuláció esetében is óva intek. (A szoftverválasztás kérdéséről ld. még Axelrod [1997].)

Mivel nincs lehetőségünk a részletekbe bocsátkozni, ezért Sefton [2000] négy kritériumát emeljük ki a szimulációs programozással kapcsolatban. Ezek:

- Pontosság (*Accuracy*),
- Sebesség (*Speed*),
- Rugalmasság (*Flexibility*),
- Robusztusság (*Robustness*).

Pontosság. E tekintetben két igen komoly problémával szembesülünk. Egyrészt a *diszkretizálási problémával*, ami abból fakad, hogy hiába beszélünk folytonos változókról, ezeket a numerikus eljárások nem tudják kezelni. A legtöbb problémát általában a folytonos idő kezelés jelenti. Például egy folytonos *differenciálegyenlet*

[19] Az összehasonlítás kedvéért 1000 számból készítettünk véletlen sorrendeket és ezt összesen 10000 alkalommal végeztük el. Az eredmény egy *Pentium III*-as processzorral rendelkező PC-n a következő volt: az első algoritmus átlagosan 520 másodpercet vett igénybe a keverésekhez, a második összesen 2 másodpercet!

megoldása során diszkretizálni kell az időt. Ezt a következőképpen tehetjük.[20] Induljunk ki az alábbi kezdeti érték feladatból:

$$\frac{dx}{dt} = f(x,t), \quad x(t_0) = x_0 \tag{2.25}$$

Használjuk az *Euler-módszert* annak érdekében, hogy diszkrét időpontokban tudjuk ábrázolni az állapotváltozónkat:

$$\begin{aligned} \frac{x(t+dt) - x(t)}{dt} &= f(x,t) \\ x(t+dt) &= x(t) + f(x,t)dt \end{aligned} \tag{2.26}$$

Ezt pedig átalakítjuk egy iterációval megadott *differenciaegyenletté*, amelyet már numerikusan is tudunk kezelni:

$$x(t+\Delta t) = x(t) + f(x,t)\Delta t \tag{2.27}$$

Figyeljük meg, hogy pontosan erről a Δt mennyiségről beszélünk a (2.5) egyenletben akkor, amikor $1/Y$-t megválasztjuk. Minél kisebb értéket adunk ennek a mennyiségnek, annál kisebb a *diszkretizálási hiba*. Azt gondolhatnánk, hogy – a futási idő rovására ugyan, de – tetszőlegesen pontossá tehetjük az eredményt kellően kicsi Δt választással. Ez sajnos nem igaz. A második komoly numerikus probléma ugyanis az, hogy a számítógép a számokat csak egy bizonyos pontossággal képes ábrázolni (tárolni).[21] Egészen pontosan szólva a numerikus eljárásoknál az asszociatív törvény sérül, vagyis a lebegőpontos aritmetikában nem igaz, hogy:

$$(a+b)+c = a+(b+c) \tag{2.28}$$

A tetszőleges finomítással azonban egyre nagyobb kerekítési hibát vétünk, hiszen egyre több számot kell összeadnunk ahhoz, hogy egy induló t_0-ból egy megadott t_n-be érkezzünk. A két hiba tehát egymás ellen dolgozik, minél pontosabb a diszkretizálásunk, annál nagyobb hibát vétünk a kerekítés miatt és fordítva.

További fontos észrevétel, hogy ugyan az asszociativitás sérül, de a kommutativitás nem, azaz $a + b = b + a$. Ennek kapcsán számos további szabály, eljárás és trükk

[20] A példát Molnár [1990] munkájából kölcsönöztük.

[21] A legtöbbször úgynevezett *lebegőpontos ábrázolást* alkalmaznak. Ez azt jelenti, hogy a helyiértékek száma fix, de a tizedes pont variábilis. Ha például 5 helyiértékkel gazdálkodhatunk, akkor ábrázolhatjuk 10000-et is és 1.0001-et is, de 10000 + 1.0001 = 10001, és nem 10001.0001.

alkalmazható annak érdekében, hogy minél pontosabb eredményeket kapjunk. A lebegőpontos aritmetikáról részletes ismertetés – és számos hasznos tipp – áll rendelkezésre Knuth [1987] könyvében. Mi itt két eljárást mutatunk be az átlag és a szórás kiszámítására. Ezek rendkívül sokszor előfordulnak a számításainkban, és jó tudni, hogy létezik egy kisebb numerikus hibát vétő számítási módszer, az amúgy megszokott formulák mellett. A következő formula x_i, $i = 1, 2, \ldots, n$ sorozat átlagát és szórását adja:

$$\begin{aligned} &M_1 = x_1, \quad M_k = M_{k-1} + (x_k - M_{k-1})/k, \quad \bar{x} = M_n \\ &S_1 = 0, \quad S_k = S_{k-1} + (x_k - M_{k-1}) \cdot (x_k - M_k), \quad \sigma_x = \sqrt{S_n/(n-1)} \end{aligned} \tag{2.29}$$

Annak ellenére, hogy a kerekítési hiba gondokat okoz, néha hasznos segítő lehet. Ilyen lehet például az, hogy egy dinamikus rendszer az instabil egyensúlyi pályákról bizonyos idő elteltével letér (ld. például: Simonovits [1998], és Balla, Benedek [1999]).

Sebesség. A rövid futásidejű, gyors eljárások készítése szintén kritikus kérdés. Sajnos hiába képes a számítógép milliárdszor gyorsabban számolni, mint az ember, az exponenciálisan növekvő számítási komplexitású feladatok esetén ez nagyságrendi előrelépést[22] nem jelent az emberhez képest. Ennek illusztrálására vegyük például a klasszikusnak számító utazó ügynök problémát.[23] Ahhoz, hogy az optimális utat megtaláljuk, minden lehetőséget végig kell számolni. Ezt az eljárást *teljes leszámolásnak* nevezzük. Ez n darab város esetében összesen $(n-1)!$ számú lehetséges utat jelent. Ha "kézzel" látunk neki a feladatnak, akkor 3-4 várost szinte ránézésre meg tudunk oldani, 7-8 városnál nagyobb feladathoz viszont hozzá sem kezdünk. Ha a számítógépre bízzuk a keresést, akkor a számítási idő 14-16 város esetében éri el azt a határt, amelyet már nehezen tartunk elfogadhatónak. A valóságban azonban sok olyan analóg feladat van, ahol a "városok" száma több száz is lehet. Úgy tűnik, a számítógép felfedezése ellenére is reménytelen ez a feladat. A számítógép azonban óriási teret enged a *visszacsatolásnak*. A számítások eredményei elraktározhatók, összehasonlításokat, rendezéseket hajthatunk végre, aminek segítségével olyan irányba mozdulunk el, ahol az optimális út megtalálásának nagyobb a valószínűsége. Természetesen feltételezhetjük azt, hogy "kézi számolás" esetén is valamiféle algoritmust követünk, ám a korábbi eredményeket tárolni, és újra átnézni a számítógépnél senki sem tudja hatékonyabban. Az algoritmusok alkalmazása nem garantálja, hogy az optimális megoldást kapjuk eredményül. Viszont minél több időt biztosítunk az algoritmus

[22] Van olyan probléma, ahol az ember és a számítógép ugyanazon a komplexitási fokon ragad meg. Például a Ramsey-féle $R(2,2)$, $R(3,3)$, $R(4,4)$ kézi kiszámítása nem jelent gondot, $R(5,5)$-öt azonban számítógép segítségével sem lehet megoldani. (ld.: Leader [2001]).

[23] Az utazó ügynök feladatban n darab várost kell egy ügynöknek úgy bejárni, hogy minden várost csupán egyszer érint, és a megtett összes út a lehető legrövidebb. Az egyes városok közti távolságok adottak.

számára, annál közelebb kerül az optimális megoldáshoz, és már viszonylag rövid idő alatt is elfogadható – az optimum közelében levő – eredményt kapunk. Látható, hogy ebben az esetben is a pontosság és a sebesség egymás ellen dolgozik, és azt a megoldást kell megtalálni, amelyik az elfogadható kompromisszum a két hatás kielégítéséhez.

Rugalmasság. Rugalmasság alatt azt értjük, hogy az elkészített eljárás különböző problémák egy viszonylag széles skáláját mennyire képes megoldani alapvető módosítások nélkül. Például a (2.5) folyamatábrán látható algoritmustervbe igen könnyű volt beilleszteni a kereskedési stratégia megváltozását. Erre a kritériumra azért van szükség, mert a szimulációs kísérletezés során olyan irányok bukkanhatnak fel, amelyekre a program készítésekor még nem számíthattunk, így a változtatások és módosítások elkerülhetetlenül megjelennek egy bonyolultabb szimuláció esetén. Az is természetes, hogy ez a kritérium – és a következő is – sokkal szubjektívebb, mint az előző kettő, ezért a rugalmasság figyelembevételére csak ajánlásokat lehet adni, konkrét módszereket nem.

Robusztusság. A robusztusság két fogalmat takar. Az első, hogy az indulóértékek nem befolyásolják a minőségi megoldást, azaz különböző indulóértékekből indulva hasonló futási időt és hasonló jelenséget tapasztalunk (pl.: konvergenciát). Másrészt ha valamilyen külső paramétert hibával becsülünk – ami közgazdasági modelleknél természetes –, akkor ez a hiba nem sokszorozódik meg a futás végére.

2.7. Verifikáció, validáció és modellkalibrálás

Elkészítettük a programot és meghatároztuk az elsődleges kísérleti értékeket is, tehát nincs más hátra, mint hogy végrehajtsuk a futtatásokat. Ennél a pontnál kell kiderülnie, hogy az általunk beprogramozott modell valóban a valóságot írja-e le, és ha nem, akkor vajon azért nem, mert rosszul programoztuk a modellt, vagy pedig azért, mert eleve helytelen volt a modellfeltevésünk. Látható, hogy az elsődleges kísérlettervezés döntő fontosságú, hiszen ezek alapján hozzuk meg a verifikáció, validáció és kalibrálás szempontjából kritikus döntéseket.

Verifikáció. A verifikációs fázisban a programozás helyességéről akarunk meggyőződni. Egy program helyes működésének ellenőrzése rendkívül bonyolult feladat és számos programozási szakkönyv foglalkozik a témával. Tökéletes verifikációs eljárás a szimulációs eljárások számára sincs, de a szakirodalom megemlít néhány fontos szabályt, melyek a következők:

- A program fokozatos futtatása, majd továbbfejlesztése,
- Kételkedés (próbáljuk megmutatni, hogy a program hibás),

- A modell részletes átvizsgálása (akár több személy által),
- Teszt-futtatások (olyan input adatokkal, amelyek speciális eredményt adnak),
- Ellenőrzés és grafikus megjelenítés (a szimuláció futás közbeni állapot-változóinak megjelenítése - *animáció*).

Megjegyezzük, hogy a pénzügyi szimuláció esetén valamennyi fenti verifikációs eszközt igénybevettük. A fokozatos programépítés keretén belül először csak a részvény-árfolyam szimulációja készült el, majd a portfólió-kezelő, ezután az opció-értékelő, majd legvégül kerültek be az egyes lehetséges kereskedési stratégiákat megvalósító részek. Ezeket a félkész programokat folyamatosan ellenőriztük, és csak akkor léptünk tovább, miután feltételeztük helyes működésüket.[24] Számos olyan teszt-futtatást is végrehajtottunk, amelyben E, r, μ, σ, és T/Y értékeit változtattuk. Figyeljük meg, hogy a $\sigma = 0$ választással teljesen determinisztikussá tehető a modell, ezért ez egy különösen jó tesztelési lehetőség volt, amellyel gyakran éltünk.

Validáció. Már a korábbiakban is hangsúlyoztuk, hogy érdemes az analitikusan kiszámítható eredményeket numerikusan megismételni, és a numerikus és analitikus eredményeket összehasonlítani. Nagyon sok esetben alkalmazható az empirikus eljárás is, azaz tényleges mintaadatokat helyettesítünk vissza a modellbe. Például egy gyártósor modellezése esetén az elsődleges kísérleteknél nem az inputadatok segítségével generált véletlenszámokkal futtatjuk a szimulációt, hanem magukkal az inputadatokkal. (Ez így már determinisztikus szimuláció). Mivel ebben az esetben az outputadatok is ténylegesen megfigyelhetők, ezért a valós és a szimuláció által generált outputadatok összehasonlítása szintén jó módszer a modell helyességének ellenőrzésére. Számos további teszt áll rendelkezésre a modell validálásához:

- Folytonosság-teszt (apró változtatás az input adatokban kis változást kell előidézzen az output adatokban),
- Érzékenységvizsgálat (változtatás a paraméterekben milyen változást eredményez az output adatokban),
- Konzisztencia-teszt (azonos futási körülmények közt zajló szimulációk hasonló eredményt kell adjanak)[25],
- Degenerálási teszt (ha a modell egy fontos elemét kiiktatjuk, akkor az outputadatokon ez meg kell, hogy jelenjen),
- Abszurd feltételek (hasonlóan a verifikációhoz, itt is speciális körülmények közötti futtatásokat értünk).

[24] A véletlenszám-generátor hibás működésére így is csak később derült fény.
[25] Azonos futási körülmények esetén csak a véletlenszám-generátor által készített véletlen sorozatok különböznek. Ehhez a generátort különböző indulóértékekből (*magból*) kell indítani.

A pénzügyi szimulációs modellnél számos validációs eljárást alkalmaztunk, így a szimulált opcióérték összehasonlítását a *Black-Scholes-féle* analitikus eredménnyel, és elvégeztük a fenti pontokban összefoglalt teszteket is.[26]

Kalibrálás. A modellkalibrálás rendkívül érdekes része a szimulációs modellezésnek. A kalibráció során a modell paraméterváltozóit kell megválasztanunk. Abban az esetben, ha ez a valós környezetből megfigyelhető és megbecsülhető, akkor a modellkalibrálás gyakorlatilag az inputanalízis része. Elképzelhető azonban, hogy egy-két paramétert nem tudunk megfigyelni, mégis nagy szerepe van a szimulációban. Ilyenkor például szakértői becslésekre – gyakorlatilag megérzésekre – kell hagyatkoznunk. Elképzelhető azonban, hogy a szimulációban szereplő objektumok – például gazdasági szereplők – valamilyen viselkedése elméletileg alátámasztható – például az optimalizálás. Ebben az esetben a szimuláció egyes részeinél a helyes paraméter beállítást egy optimum-számítási feladattal számíthatjuk ki, azaz a következő feladatot oldjuk meg:

$$\max_K f(K,x), \tag{2.30}$$

ahol f reprezentálja a szimulációs folyamatot, K pedig a kalibrálni kívánt paramétert. Az is elképzelhető, hogy nem ismerünk ilyen magatartási formát, de meg tudjuk figyelni a modell valamely outputváltozóját. A szimulált és valóságos outputadatok ($o(x)$) összehasonlítása segítségével szinten kalibrálhatunk egy paramétert[27]:

$$\min_K \left(f(K,x)-o(x)\right)^2. \tag{2.31}$$

A (2.30) és (2.31) feladatok megoldása egyáltalán nem egyszerű, hiszen f-ről sokszor semmit sem állíthatunk – néha még azt sem, hogy az függvény. A szimulációk numerikus optimalizálása a későbbiekben még többször is elő fog kerülni.

[26] A konzisztencia-teszt során fedeztük fel a hibás véletlenszám-generátor problémáit, és ekkor implementáltuk a *gasdev* eljárást a kizárás módszere helyett.

[27] Illusztráció céljából képzeljünk el egy telefonos *Call-Center*-t, ahol nincs információnk arról, hogy egy olyan ügyfél, akinek túl sokat kellett várakoznia az operátorra és ezért kiszolgálás előtt letette a telefont, milyen valószínűséggel hívja újra egy adott időn belül a központot. Kiindulunk egy hipotetikus értékből és összehasonlítjuk, hogy a szimulált eredmények mennyire egyeznek a valós értékekkel, majd ezt addig módosítjuk, amíg a kívánt pontosságot el nem érjük.

2.8. Kísérletezés

Ha sikerült meggyőződni a modell helyességéről, el lehet kezdeni a kísérletezést. Ebben a fázisban számítunk új eredményekre és értékes összefüggések feltárására. Természetesen ez az a pont, amely leginkább függ az adott problémától és a kísérletezőtől, így nem igazán lehet általános módszereket megfogalmazni. Egyedül az fontos, hogy a kísérletezési fázis során sokszor vissza-vissza kell lépnünk a modellépítés és validáció/verifikáció lépésekhez, mivel az új kísérletek újabb és újabb elemek beépítését igényelhetik a szimulációs programba. Ezekről a lépésekről nem szabad megfelejtkezni, hiába volt validált egy-egy korábbi kísérlet.

2.9. Output analízis

Végezetül szót kell ejtenünk arról, hogy az output adatokat részletesen elemezni kell annak érdekében, hogy a várt összefüggéseket statisztikailag is alátámaszthassuk. Ehhez ugyanazok az eszközök nyújtanak segítséget, mint amelyeket az inputanalízisnél már felsoroltunk, ezért az elemzési módszereket nem részletezzük. Ehelyett arra a problémára összpontosítjuk a figyelmünket, hogy hogyan lehet meghatározni a helyes replikációszámot, azaz a *mintaméretet*. Nyilvánvaló, hogy minél nagyobb a mintánk, annál pontosabb lesz az output változónk, viszont a sokszori replikálás nagyon időigényes feladat. Problémát jelent az is, hogy a legtöbb szimuláció esetében nem lehet statisztikai adatot gyűjteni addig, ameddig el nem telik egy ú.n. *felmelegedési periódus* (*warm-up period*). A felmelegedési periódus alatt azt az időszakot értjük, ameddig a modell egyes elemei a valóságtól eltérően működnek, mert induláskor egy teljesen üres rendszert kezdtünk szimulálni. (Például egy napi 24 órás gyártósor esetén nagyon ritka, hogy minden futószalag üres, minden gép hibamentes, minden erőforrás szabad, minden alapanyagból kellő mennyiség áll rendelkezésre, stb.) Hogy számítási időt takarítsunk meg, elképzelhető, hogy több replikáció helyett egy nagyon hosszú futtatás felmelegedési periódusa utáni eredményeit vizsgáljuk, és azokat osztjuk fel több részre. A felosztásnál azt kell szem előtt tartani, hogy elegendő elem kerüljön a mintába, ugyanakkor elég hosszúak legyenek a mintaperiódusok ahhoz, hogy az egyes mintákat függetlennek lehessen tekinteni egymástól. Ezt az eljárást a szakirodalom *batch-mean* módszernek nevezi.

A futási időt más módszerrel is csökkenthetjük. Ezt a módszert akkor alkalmazzuk, amikor csak a kimeneti valószínűségi változó várható értékét szeretnénk meghatározni. Erre a becslésre triviálisan a mintaátlagot használtuk, most azonban bemutatunk egy hatékonyabb módszercsaládot, a *varianciacsökkentő módszereket*. A feladat tehát az, hogy olyan becslési eljárást adjunk, ami – lehetőleg torzítatlanul – kisebb varianciával becsüli a várhatóértéket, mint az egyszerű mintaátlag.

Avramidis-Wilson [1996] ad jó áttekintést a szimulációs modellezésben alkalmazott varianciacsökkentő módszerekről. Általában kétféle módszer létezik:

(i) korrelált minta készítése, amely hatásosabb becslést adhat, mint a FAE minta;
(ii) becslés a paraméterek feltételes eloszlásából, mikor a feltételben szereplő változók eloszlásának elméleti paraméterei ismertek (pl. kontroll változók módszere). Ekkor a becslés hatásossága a korreláció mértékétől függően javítható.

Megjegyezzük, hogy a szimulációs pénzügyi modellben az árfolyammozgás játszhatja a kontroll változó szerepét. Egydimenziós esetben célszerű a futamidő végi részvényárfolyamot használni. Ez korrelál az opció árával és ismertek az elméleti paraméterei. Nézzük tehát a kontrollváltozók módszerét!

Tegyük fel, hogy találunk egy olyan valószínűségi változót a modellben, amelynek elméleti paraméterei ismertek, és erős lineáris korrelációban van a modell eredményváltozójával. Ekkor használhatjuk az alábbi lineáris becslő formulát:

$$Y_{CV} = Y - \beta\left(C - \mu_C\right), \tag{2.32}$$

ahol Y a kimeneti változó értéke, Y_{CV} a kontrollváltozóval korrigált értéke, C a kontrollváltozó, amelynek elméleti várható értéke ismert, éspedig μ_C. (Több kontrollváltozó esetén ez utóbbi két változó értelemszerűen vektor, így a fenti kifejezés jobb oldalán egy skaláris szorzás szerepel.) A kontrollváltozóval korrigált változó becslése nyilvánvalóan torzítatlan, és hatásossága a β paraméter értékétől függ, ugyanis:

$$\mathrm{E}\left(\overline{Y}_{CV}\right) = \mathrm{E}\left(\overline{Y}\right) - \beta\,\mathrm{E}\left(\overline{C} - \mu_C\right) = \mathrm{E}\left(\overline{Y}\right). \tag{2.33}$$

Nyilván akkor hatásos a becslés, ha tényleg csökken a korrigált változó varianciája. Anderson [1958] belátja, hogy β optimális értékét – az *optimális kontrollt* – a következő formula adja:

$$\beta = \frac{\sigma_{YC}}{\sigma_C^2}, \tag{2.34}$$

ahol σ_{YC} az eredeti kimeneti változó és a kontroll változó kovarianciája, σ_C^2 pedig a kontroll változó szórásnégyzete. (A továbbiakban σ-val mindig az elméleti, *s*-sel mindig a tapasztalati szórást jelöljük). Nyilvánvaló, hogy ebben az esetben annál hatásosabb a becslés, minél nagyobb az eredeti változó és a kontrollváltozó közti korreláció, és minél kisebb a kontrolváltozó szórása, ugyanis

$$\mathrm{Var}(Y_{CV}) = \sigma_Y^2 + \beta^2\sigma_C^2 - 2\beta\sigma_{YC}\,. \qquad (2.35)$$

A becslés akkor hatásos, ha $\beta^2\sigma_C^2 - 2\beta\sigma_{YC} \le 0$. Így a (2.34) felhasználásával azt kapjuk, hogy

$$\frac{\sigma_{YC}^2}{\sigma_C^2} - 2\frac{\sigma_{YC}^2}{\sigma_C^2} \le 0\,, \qquad \text{azaz} \qquad \frac{\sigma_{YC}^2}{\sigma_C^2} = \rho_{YC}^2 \ge 0\,. \qquad (2.36)$$

Mivel ρ_{YC} a (többszörös) korrelációs együttható, ezért látható, hogy a variancia csökkentés feltétele a becsült és a kontrollváltozó(k) közti korreláció. Ez az elméleti eset, a gyakorlatban a szabadságfokok veszteséget indukálnak, ezért az alábbi, szabadságfokok által meghatározott küszöbértéket kapjuk:

$\rho_{YC}^2 > \frac{q}{n-2}$, ahol q a kontrollváltozók száma, n a minta mérete.

Mivel β nem ismert, ezért mintából kell becsülni. Ekkor $\hat{\beta} = b = \frac{s_{YC}}{s_C^2}$. Sajnos ebben az esetben a becslés torzítatlansága már csak akkor garantált, ha (Y,C) együttes eloszlása normális[28]. Vagyis csak ekkor igaz, hogy

$$\mathrm{E}\left(\frac{s_{YC}}{s_C^2}(\overline{C}\text{-}\mu_C)\right) = \mathrm{E}\left(\frac{s_{YC}}{s_C^2}\right)\mathrm{E}(\overline{C}\text{-}\mu_C) = 0\,. \qquad (2.36)$$

Az általunk készített pénzügyi modellben sem Y (opcióárfolyam) sem pedig C (részvényárfolyam a futamidő végén) nem követ normális eloszlást, így együttes eloszlásuk sem lehet normális eloszlás. C-ről viszont tudjuk, hogy lognormális eloszlású, vagyis a logaritmusa normális eloszlású. Y elméleti eloszlása nem ismert, a grafikus megjelenítések alapján K[29] bizonyos értékeire lognormális eloszlás sejthető (ez a $K < 20$ tartomány).

A kontroll változók módszerének általánosításairól nem normális eloszlások esetére Nelson [1990] ad áttekintést, melyben lognormális eloszlásra konkrétan nem mutat be módszert.

---

[28] Ez abból a tételből következik, hogy a normális eloszlás átlag vektora és variancia-kovariancia mátrixa függetlenek. Ld. pl.: Móry, Székely [1986] vagy Anderson [1958].

[29] K jelentette a befektetési stratégia szabad paraméterét a szimulációs modellben.

Modellünkben megoldható, hogy úgy transzformáljuk a változókat, hogy azok már normális eloszlást kövessenek. Legyen X az opció ára és definiáljuk most Y-t, mint X logaritmusát, vagyis legyen $Y = \ln X$, továbbá jelölje C már eleve az árfolyamok logaritmusát. Így Y is és C is normális eloszlású változók[30], vagyis annak szükséges feltétele, hogy (Y,C) együttes eloszlása normális, teljesül. Képezzük az alábbi várható érték becslést[31]:

$$X_{CV} = X - b(C - \mu_C), \quad \text{ahol} \quad b = \frac{s_{YC}}{s_C^2}, \tag{2.36}$$

ahol X lognormális eloszlású és $Y = \ln X$. Mivel $\mathrm{E}\left(\frac{s_{YC}}{s_C^2}\left(\overline{C} - \mu_C\right)\right) = 0$ most is igaz, ezért $\mathrm{E}(\overline{X}_{CV}) = \mathrm{E}(\overline{X})$, vagyis a becslés továbbra is torzítatlan. Az persze nem biztos, hogy β most is az optimális kontroll, ugyanis:

$$\begin{aligned} &\mathrm{Var}(X_{CV}) = \sigma_X^2 + \beta^2\sigma_C^2 - 2\beta\sigma_{XC}, \\ &\frac{\sigma_{YC}^2}{\sigma_C^2} - 2\frac{\sigma_{YC}}{\sigma_C^2}\sigma_{XC} \le 0, \\ &\frac{1}{2} \le \frac{\sigma_{XC}}{\sigma_{YC}}. \end{aligned} \tag{2.37}$$

Ezek szerint varianciacsökkenés abban az esetben várható, ha legalább fele akkora (X,C) kovarianciája, mint (Y,C)-jé. Belátható, hogy a varianciacsökkentés akkor maximális, ha a fenti hányados éppen egy, azaz $\sigma_{YC} = \sigma_{XC}$. Ugyanakkor független valószínűségi változók függvényeire vonatkozó tétel alapján használható a (2.36) becslésnél az optimális kontroll, ami így $b = \frac{s_{XC}}{s_C^2}$.

2.12. Tétel: *A* (2.36) *becslés ez utóbbi b választás esetén is torzítatlan marad.*

[30] Feltéve persze, hogy igaz X lognormalitására vonatkozó hipotézisünk.

[31] Az ökonometriában az ilyen tipusú feladatokra semi-log modelleket illesztenek, vagyis $Y = \ln X$-re írnak fel lineáris összefüggést valamilyen magyarázó változókkal (esetünkben kontrollváltozókkal). Könnyen belátható, hogy ezzel a modellel X várható értékét torzítottan becsüljük.

A tétel azért érdekes, mert a szakirodalomban nem található meg, annak ellenére, hogy pénzügyi szimulációs alkalmazása rendkívül jelentős.[32]

Bizonyítás: *Tudjuk, hogy* s_{YC}, s_C^2 *és* $\overline{C}$ *függetlenek egymástól, ekkor azonban* $f(s_{YC}), s_C^2$ *és* $\overline{C}$ *is függetlenek egymástól, ahol f folytonos, szigorúan monoton transzformáció. Viszont* s_{YC} *és* s_{XC} *kapcsolatát pont egy ilyen f leképzés írja le. Gondoljuk meg ugyanis, hogy az empirikus kovariancia valójában egy leképzés az* $Y^n \times C^n$ *mintatérből a valós számegyenesre (, ahol n a mintaméret). Az* s_{XC} *leképzés esetében pedig az n darab Y tengely helyett azok szigorúan monoton transzformációját,* $X = \mathrm{e}^Y$ *tengelyeket használjuk. Így tehát a két leképzés szintfelületei bijektív viszonyban állnak egymással. Nyilvánvalóan a fenti gondolatmenet bármilyen olyan eloszlás esetében használható, mely szigorú monoton transzformációval megkapható a normális eloszlásból.*

A (2.36) helyett tehát használhatjuk az alábbi szintén torzítatlan, hatásosabb becslést:

$$X_{CV} = X - \frac{s_{XC}}{s_C^2}\left(C - \mu_C\right). \tag{2.38}$$

Ezzel a varianciacsökkentő módszerek tárgyalását befejeztük, de megjegyezzük, hogy számos hatékony eljárást találhatunk Boyle et al. [1997], Hull, White [1988] cikkében.

2.10. Összefoglalás

Ebben a fejezetben bemutattuk azokat a lényeges elemeket, eljárásokat és bizonyításokat, amelyek egy általános szimulációs modell elkészítésénél a legfontosabb szerepet játsszák. E módszerek segítségével elsősorban mikroökonómiai, vállalat-gazdasági és pénzügyi szimulációkat tudunk megvalósítani, azonban komolyabb közgazdaságelméleti modell kezeléséhez még számos kérdés tisztázatlan. Hogyan lehet közgazdasági szereplők viselkedését modellezni? Milyen módszerekkel lehet egy szimulációt optimalizálni? Ezekre a kérdésekre igyekszünk válaszolni a harmadik fejezetben.

[32] Boyle et al. [1997] nagy jelentőségű cikkében is elköveti azt a hibát, hogy torzítatlan és torzításos varianciacsökkentő eljárásokat hasonlít össze.

3. Fejezet

Evolúciós módszertan

A közgazdaságtan általában az összes gazdasági szereplőt haszonmaximalizáló, optimalizáló, vagy legalábbis döntéshozó és adaptációs képességgel rendelkező szereplőnek tekinti. A legtöbb modell eleve feltételezi, hogy a szereplők az optimális viselkedést az összes rendelkezésére álló információ alapján, analitikusan meg tudják határozni, és mindvégig ezt alkalmazzák. A valóságban azonban a környezet állandóan változik. (Már az is változást okozhat, ha a szereplők nem egyszerre hozzák meg – amúgy optimális – döntéseiket.) A szereplők a valóságban állandóan *tanulnak*, *alkalmazkodnak*, *fejlődnek*. Az evolúciós eljárások ezeket a folyamatokat szimulálják külön jelentőségük van a közgazdaságtudományban.

A szimulációs modellezés során két minőségben találkozhatunk evolúciós eljárásokkal. Egyrészt – magától értődően – a modell szereplőinek viselkedését kívánjuk ezzel a módszerrel „okosabbá" és realisztikusabbá tenni. Másrészt ezek az eljárások jól szolgálhatnak a szimulációs modell egészére vonatkozóan is, amikor a modellező szerepét vállalják át. (Ahogy ugyanis egy függvény szélsőérték-helyét kereshetjük evolúciós módszerekkel, úgy ugyanez alkalmazható egy szimulációs modell optimalizálására is). Így előfordulhat az, hogy egymásba ágyazva több evolúciós eljárás jelenik meg egy-egy szimulációs alkalmazásban.

A fejezet a továbbiakban öt alfejezetből áll. Az első részben az evolúciós elméletek általános tulajdonságait tárgyaljuk. A történeti összefoglaló és az elméleti ismertetés mellett bemutatunk néhány konkrét alkalmazást, különös tekintettel a közgazdasági alkalmazásokra. A második részben a numerikus optimalizálás, a harmadikban a *genetikus algoritmus*, a negyedik részben pedig a *neurális háló* részletesebb ismertetésével foglalkozunk. Végül összefoglaljuk a fejezetet.

3.1. Evolúciós elméletek

Ma már számos evolúciós módszerrel, algoritmussal és alkalmazással találkozhatunk a szakirodalomban. Ezek rendszerint abban különbözhetnek egymástól, hogy milyen evolúciós modell szolgál az algoritmus alapjául, vagy akár ugyanolyan modell esetén, milyen mechanizmus valósítja meg a modellt. Minden modellben közös azonban az, hogy valamilyen szinten a természet optimális kiválasztódási folyamatát próbálják utánozni.

Mit csinál a természet? Tegyük fel, hogy adott egy populáció számára valamilyen kezdeti állapot. Tételezzük fel azt is, hogy ez az állapot optimális a populáció és a természet számára, például nem hal ki vagy nem szaporodik el túlságosan a populáció. Ebben az állandósult állapotban nincs fejlődés, nincs tanulás és tökéletes az egyensúly, abban az értelemben, hogy nincs semmilyen erő, amely kimozdítaná ebből az állapotból. A következő lépésben azonban úgy változtatjuk meg a populáció környezetét, hogy ez valamilyen negatív hatással van rá. A populáció egyes egyedei elpusztulnak, mások életben maradnak. A következő generációba az életben maradtak utódai kerülnek, de azok is szembesülnek a negatív hatással; elpusztulnak vagy életben maradnak, szaporodnak, stb. Az egyedek természetesen nem egyformák, egyes tulajdonságokban különböznek egymástól. Ezen tulajdonságok azonban meghatározzák, hogy a negatív hatással szemben mennyire ellenállóak. Az adott negatív hatást jobban „közömbösítők" életben maradnak és képesek továbbörökíteni ezt a tulajdonságot. Sőt, a populáció akkor sincs feltétlenül halálra ítélve, ha ez a tulajdonság (vagy ezek a tulajdonságok) a kezdeti generáció egyetlen egyedében sem fordult(ak) elő. Lehetőség van ugyanis a mutációra, azaz olyan öröklődésre, ahol az utód nem minden tulajdonsága vezethető vissza szülei, nagyszülei stb. tulajdonságaira, hanem vannak véletlen megváltozások is. Az adott feltételek mellett, számos generáció után újra létrejön az egyensúly, és ez az új állapot (tulajdonság-összetétel) lesz optimális.

Tekintsük ezt a problémát egy hagyományos optimum-feladatnak, amelyben az egyedek különböző tulajdonságai adják a lehetséges megoldások halmazát. (Minden egyed egy-egy lehetséges megoldás.) Minden lehetséges tulajdonsághoz a természet "megmondja", mennyire jók az adott egyed túlélési esélyei, azaz adott egy, az összes lehetséges tulajdonság halmazán értelmezett függvény. A populáció az utolsó generációban megtalálja a függvény maximumát. A folyamatban az a különös, hogy anélkül találja meg az optimumot, hogy ismerné a célfüggvény alakját (tulajdonságait). (Hozzátesszük, hogy ezzel a fejlődési stratégiával viszonylag gyorsan találja meg a populáció a számára optimális állapotot. Ennek azért van nagy jelentősége, mert ismeretlen célfüggvény esetében a legegyszerűbb megoldás az összes lehetséges eset kipróbálása és az optimális megoldás kiválasztása. Ez azonban óriási számítási időt vehet igénybe.) E szempont miatt érdekes számunkra az evolúciós eljárás, hiszen – mint a bevezetőben elmondtuk – olyan problémákra keressük a megoldást, amelyben a célfüggvény nagyon bonyolult, vagy akár nem ismert tulajdonságú.

Feltehetjük a következő kérdést: Nem véletlenül találta-e meg a populáció az optimális tulajdonságokat? A válasz igen, a véletlennek óriási szerepe van fejlődésben. Viszont ez a véletlen nem "vak". Nem össze-vissza bolyongunk egy számunkra teljesen ismeretlen helyen, hanem fejlődési irányokat, utakat derítünk fel. Az útról persze többször is letérünk, de a lényeg az, hogy minden következő lépésünket valamilyen módon az előző lépések helyessége befolyásolja. (Az, hogy az algoritmus optimumba jut az nem véletlen

szerű, de hogy melyik egyed éri el az igen) Azt lehet tehát mondani, hogy ezekben az algoritmusokban a tanulás vagy fejlődés irányított.

Ez a megközelítés az evolúciós eljárások egyik fő iránya. Célja az optimalizálás, az algoritmust pedig *genetikus algoritmus* néven nevezi a szakirodalom. Természetesen sok minden még nincs tisztázva; például hogyan zajlik az öröklődés, hány egyed van egy populációban, hány generáción keresztül figyeljük meg a populációt, stb. Ezekre a kérdésekre a 3.3. alfejezetben adunk választ.

Nézzük most tovább az evolúciós eljárásokat! Sikerült találni egy olyan eljárást, amely egy diszkrét változásra egy hosszú folyamat után optimális választ ad. Sok esetben azonban nagyon gyakori a változás és nagyon rövid idő áll rendelkezésre ahhoz, hogy a változásra a helyes választ megadjuk. Azért nehéz a gyors válasz, mert nem ismerjük azt a függvényt, amely minden egyes környezeti állapothoz megadja az optimális (vagy megfelelő) megoldást. A természetben rengeteg olyan eset fordul elő, amikor nagyon gyorsan kell válaszreakciót adni. Ilyen például az arcfelismerés, ahol a szemünkbe érkező nagy mennyiségű képinformációt kell valamilyen függvénynek feldolgoznia, és végeredményül az adott személyt meghatároznia. A természetben az ilyen típusú függvények keresése és állandó módosítása, javítása a *tanulás*. Így kézenfekvő volt, hogy a modellezéshez az agy működését (neuronok működését) használták. A *neurális háló* tehát olyan tanuló algoritmus, amely nagy mennyiségű példa alapján egy adott bemeneti-halmaz és kimeneti-halmaz közti összefüggést próbálja megtalálni. Sok esetben az előrejelző modellek készítésénél is a fenti problémával szembesülünk. Nagy mennyiségű adattal rendelkezünk, de az adatok között bonyolult és sokrétű összefüggés lehet és a modellezést az adatok zajossága is bonyolíthatja. Ebben az esetben az összefüggések keresését érdemes a neurális hálóra bízni. A neurális hálóval részletesen a 3.4. alfejezetben foglalkozunk.

Végül meg kell jegyeznünk, hogy bár a genetikus algoritmus és a neurális háló az evolúciós elméletek két talán legfontosabb pillére, rengeteg egyéb evolúciós módszer létezik és nagyon sok "vegyes" eljárás található, mind az algoritmusok, mind az alkalmazások szintjén. Így például a neurális háló nem csak függvénykeresésre, hanem optimalizálásra is alkalmazható, a genetikus algoritmus felhasználható a neurális háló hierarchiájának (struktúrájának) kialakításánál, hagyományos optimalizáló módszereket (pl. gradiens-módszer) és heurisztikus optimalizáló eljárásokat (pl. *szimulált lehűtés – simulated annealing*) kapcsolhatunk össze a genetikus algoritmussal, stb.

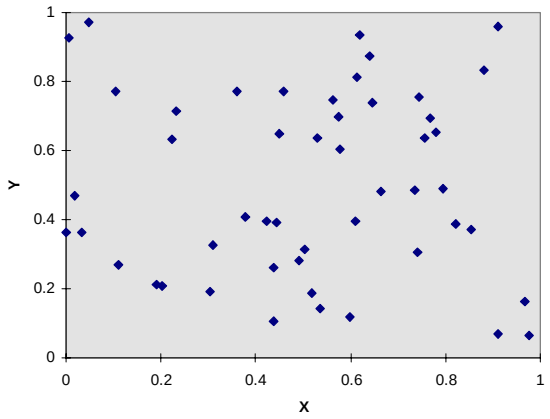
Röviden bemutatunk néhány tényleges, illetve lehetséges alkalmazást. Ezek a példák szemléltető célt szolgálnak, és nem törekszünk sem a teljességre, sem pedig az alkalmazások valamilyen csoportosítására, klasszifikációjára. Nézzünk először két klasszikus – ám nem közgazdasági – alkalmazást.

A számítógéppel dolgozók régi nagy álma, hogy a számítógépet nem kézzel, hanem hanggal tudják irányítani, diktálni tudjanak a számítógépnek, azaz a gép ismerje fel a beszédet. A feladat több szempontból is nehéz. A beérkező hang nagyon sok információt tartalmaz, ebből nem mindenre van szükség a beszéd felismeréséhez. Nehéz azonban eldönteni, hogy mit lehet kiszűrni vagy a hatását tompítani. A betűket egyenként felismerni lehetetlenség, de sokszor az élőbeszédben a szavakat is összemossuk. Sok szónak más az írása, mint a kiejtése. Minden ember egy kicsit máshogyan beszél, más hangmagasságon, más tempóban, más kiejtéssel. A nehézségeket még hosszan sorolhatnánk. A megoldást többek között neurális hálók alkalmazásával keresik. A feladat egyébként kifejezetten neurális háló típusú feladat, hiszen analóg az emberi agy beszédfelismerési mechanizmusával. A neuronoknak azt a "leképzést" kell megtalálniuk, amely minden beérkező hangmintára a megfelelő írott szöveget adja eredményül. A számítástechnika jelenlegi lehetőségei még nem oldották meg a „tökéletes" beszédfelismerést, de részsikerek már vannak. Ez azt jelenti, hogy háttérzaj-mentes környezetben, jól tagolt, meghatározott – maximum néhány száz szavas – szókincsből álló beszédet a számítógép akkor képes felismerni, ha a beszélő megfelelő mennyiségű mintával megtanította a saját beszédstílusára a neurális hálózatot. A fentihez kissé hasonló írott szöveg felismerését azonban gyakorlatilag már sikerült megoldani, és az ezen feladatot megoldó programok már rendelkezésére állnak. (Részletes ismertetés található: Institute of [1989] és Holmes [1993]).

A genetikus algoritmus alkalmazására számtalan példa található. Ezek közül egy példa az automata sebességváltó. A feladat az, hogy optimális időpontokban kapcsoljunk egy-egy fokozattal előre gyorsításkor, azaz úgy váltsunk sebességet, hogy mondjuk 100 km/h-ra a lehető legrövidebb idő alatt gyorsuljunk fel. Bár az optimalizálandó függvény formája igen bonyolult – hiszen az adott fokozatok nyomatékgörbéi erősen függenek attól, hogy mekkora sebességnél történik a kapcsolás – a genetikus algoritmus számára könnyen és gyorsan megoldható ez a probléma.

Nézzünk néhány alkalmazást közgazdasági területről! Elsőnek az utazó ügynök problémát. Egy példafeladatban 50 várost (ld.: 3.1 ábra) kell a lehető legrövidebb útvonalon bejárni (ld.: Pham-Karaboga [1998]). Ha az optimális megoldást keressük, akkor 49!/2 (azaz több mint $3 \cdot 10^{64}$) lehetőséget kellene végigszámolni. Ez még a mai leggyorsabb számítógépek számára is kivárhatatlanul hosszú feladat. A jelen esetben mind a genetikus algoritmus, mind a neurális háló alkalmazható a legrövidebb út megtalálásához. Mindkét algoritmus nagyon rövid – és közel azonos – idő alatt ad eredményt. A genetikus algoritmus által megtalált legrövidebb út hossza 5.58 egység, a neurális hálóé 6.61 egység volt. Jelen esetben tehát a genetikus algoritmus pontosabb eredményt adott. Nem lehetünk azonban biztosak abban, hogy a megtalált 5.58 egység hosszúságú út a legrövidebb, csak azt tudjuk, hogy igen jó – optimumhoz közeli – megoldást kaptunk. Az egyes algoritmusok maguk is "tudják", hogy nem biztos, hogy

optimális megoldáshoz jutottak, ezért a keresés, illetve a tanulás leállásának különböző kritériumokat adhatunk.



3.1. ábra: Térkép (50 város)

A feladat kicsit módosított változatát több szállítással foglalkozó vállalat is alkalmazza. Egy olajtársaság számára például fontos kérdés, hogy meghatározott számú telephelyeiről milyen módon juttatja el a benzint a kutak számára. Itt nem egyetlen "ügynökről" van szó, hanem többről, hiszen a társaságnak több olajszállító kamion áll rendelkezésére. Ezek átlagos sebessége és tárolókapacitása is különböző lehet. A feladat az, hogy a telephelyekről a lehető legkisebb költséggel szállítsák az üzemanyagot a kutakhoz. A problémát genetikus algoritmus segítségével oldották meg (ld.: Himanen [1998]).

A következő alkalmazást mobil-telefon társaságoknál fejlesztették ki. A probléma az, hogy a kommunikációhoz szükséges rádió adó-vevő központokat milyen sűrűn és hova helyezzék el. Ha egy körzet nincs lefedve (nincs térerő), akkor onnan nem lehet telefonálni, ha pedig nincs elég sűrűn lefedve – megfelelő kapacitással –, akkor csúcsidőben nem lehet telefonálni (*túlterhelés*). Látható, hogy a feladat különösen bonyolult, hiszen nem csak a földrajzi adottságokat kell figyelembe venni, hanem a lehetséges használat mértékét is. A problémát többek között neurális hálózattal és genetikus algoritmussal oldották meg.

Az ökonometriai módszertani kutatások során is rendszeresen alkalmaznak genetikus optimalizálást, például a nemlineáris regressziós modellek, illetve a becsléselmélet *loglikelihood* függvényeinek optimalizálása esetén. Ilyenkor ugyanis a célfüggvény nagyon bonyolult, és sok lokális optimummal rendelkezik, így a hagyományos optimalizáló algoritmusokat nem lehet használni.

Konkrét alkalmazásokat elsősorban szimulációs modellekben láthatunk (pl.: világmodell). Ilyen modellekben a szimulációt *fekete-doboz* (*black-box*) függvénynek, a szimuláció egyes paramétereit exogén – kívülről szabályozható – változónak tételezzük fel (pl.: nyersanyagok kitermelésének mennyisége), míg más változókat a szimuláció eredményváltozóinak tekintünk (pl.: népességnövekedés, bioszféra állapota, stb.). A szimuláció mindig egy adott paraméter-beállítás mellett ad választ bizonyos kérdésekre (pl.: ha az őserdők fakitermelésének szintjét 20%-kal csökkentjük, akkor az átlagos GDP növekedés 0.5%-ponttal csökken). A szimuláció optimalizálása esetén mi definiálhatjuk az exogén változóinkat, azok lehetséges értékét, és mi dönthetjük el, hogy mely eredményváltozók milyen értékét tartjuk kívánatosnak. (Pl.: milyen legyen az egyes nyersanyagok kitermelésének a mértéke annak érdekében, hogy meghatározott idő alatt a GDP legalább 1%-ponttal növekedjen és a környezetszennyezés mértéke ne növekedjen.) Szimulációk optimalizálása tipikusan a genetikus algoritmus számára kitalált feladat. Ráadásul a genetikus algoritmus képes több célfüggvény esetén az efficiens megoldások meghatározására is (ld.: Benedek [1999b]).

Végezetül nézzünk egy példát olyan előrejelzési modellre, amelyet rendszerint bankok alkalmaznak. A feladat az, hogy a bank által üzemeltetett ATM bankjegy-automatában optimális szinten legyen az adott bankjegyek mennyisége és választéka. Az optimális szint itt azt jelenti, hogy ne legyen nagyon sok, mert ennek napi kamata költségként jelentkezik az adott banknál, de ne is legyen nagyon kevés, mert a kifogyó automata erősen rontja az adott bank image-ét. Továbbá, ha túlságosan gyakran kell feltölteni az automatát, akkor a szállítási költség lesz nagyon magas. Pénzváltó automaták esetében különösen fontos és nehezen megválaszolható kérdés az, hogy az egyes valuta-fajtákból mekkora összeg legyen raktározva. A feladatot neurális háló és genetikus algoritmus segítségével lehet kezelni. A neurális hálózat segítségével, a múlt tapasztalatai alapján meg lehet határozni az adott bankjegy-automata várható forgalmát (akár mint valószínűségi változót). Ezután a genetikus algoritmus segítségével meghatározzuk az optimális feltöltési stratégiát. Az optimalitás itt megint több szempontot érinthet, ugyanis nem biztos, hogy a legalacsonyabb üzemeltetést kell választani akkor, ha a véletlen hatások miatt ez nagyon kockázatos (gyakori kifogyás).

Végezetül összefoglaljuk azokat a feltételeket, amelyek megléte ajánlott abban az esetben, amikor evolúciós eszközökkel próbáljuk megkeresni a megoldást:

Analógia: Előnyös, ha az adott problémához, melyet evolúciós eljárásokkal kívánunk megoldani, található valamilyen analóg evolúciós példa, illetve a szereplők cselekvése ezzel az elmélettel magyarázható. Ilyen lehet a mintafelismerés, a fejlődés, a tanulás. A legtöbb közgazdasági alkalmazásban ez a feltétel adott.

Komplexitás: A megoldandó feladatnak elegendően bonyolultnak kell lenni ahhoz, hogy érdemes legyen evolúciós eljárásokat alkalmazni. Nagyon sok egyéb módszer ismert, és mint megállapítottuk, az evolúciós algoritmusok nem mindig pontosan a legjobb megoldást adják, hanem az adott feltételek között csak „elég jót". Egyszerűbb feladatok esetén érdemesebb matematikai és statisztikai módszereket vagy teljes leszámlálást alkalmazni.

Adat: Az evolúciós eljárások legsikeresebben akkor alkalmazhatók, ha már "elveszünk az adatok között", azaz rendkívül nagy mennyiségű és nehezen átlátható adat birtokában kell valamilyen döntési modellt meghatároznunk. Természetesen az adatoknak relevánsnak, többé-kevésbé teljesnek és hibamentesnek kell lenniük az adott feladatra vonatkozóan. Ha nincs összefüggés, azt az evolúciós eljárások általában szintén meg tudják állapítani, viszont ha hibás és/vagy hiányos adatbázissal dolgozunk, akkor az algoritmusok könnyen félrevezetődhetnek és hibás következtetéseket vonhatnak le.

Külső információ: Az evolúciós eljárásokat nem szabad olyan esetekben alkalmazni, amikor egyéb információk lehetővé teszik egyszerűbb módszerek alkalmazását. (Például ha ismert – vagy feltételezzük, hogy ismert – a modellforma és csak annak paramétereit akarjuk becsülni, vagy ha célfüggvényről tudjuk, hogy egyetlen lokális (=globális) optimális pontja létezik.) Az evolúciós eljárásokkal akkor érdemes próbálkozni, ha nem tudunk vagy nem akarunk hipotéziseket megfogalmazni az adott problémára vonatkozóan, továbbá az evolúciós algoritmusok eredményeit, mint automatikusan – a számítógép által – megfogalmazott hipotéziseket kell kezelni, s ezeket további statisztikai-matematikai módszerekkel érdemes ellenőrizni.

3.2. Numerikus optimalizálás

A genetikus algoritmust tehát azért kívánjuk bevezetni, hogy a szimulációban szereplő egyedek külön-külön optimális viselkedést legyenek képesek produkálni, vagy pedig magát a szimuláció kimenetelét szeretnénk egy számunkra optimális értékre beállítani. (Az első esetre példa, ha egy cseregazdaságot képzelünk el, ahol minden szereplő a saját hasznosságát igyekszik maximalizálni, a második esetre pedig példa lehet egy telefonos ügyfélszolgálat (*Call-Center*), ahol az operátorok számát szeretnénk minimalizálni, adott kiszolgálási idő mellett.) Azaz egy függvény szélsőérték-helyét kell megkeresnünk, ahol a függvény maga a szimuláció, vagy a szimulációban egy kisebb – önállóan működő – egység. Vegyük észre, hogy a függvényoptimalizálással ellentétben itt nem ismerjük a függvény alakját, csak behelyettesítési értékékeit, ráadásul azokat is igen korlátozott számban, hiszen minden egyes függvénykiértékelés – szimuláció – „költséges". Ez a költség a futásidő. A tényleges optimum meghatározásának így egyedül egy biztos módja van, a teljes leszámlálás – azaz az összes lehetséges pontban

való kiértékelés –, és ráadásul még ez sem biztos, a számítógép kerekítési korlátjai miatt.[1] Néha elképzelhető hogy a szimulációra vonatkozólag is tudunk valamilyen tulajdonságot bizonyítani (Lipsitz-folytonosság, folytonosság, konvexitás, stb.) vagy legalább néhány futtatás eredményeképpen feltételezni. Minél többet tudunk, illetve feltételezünk, annál hatékonyabb – gyorsabb, pontosabb – algoritmust alkalmazhatunk. A genetikus algoritmust akkor célszerű bevetni, amikor nincs semmilyen előzetes feltételezésünk a vizsgált rendszerről.

Mielőtt azonban rátérünk a genetikus algoritmusra, megmutatjuk, hogy milyen más módszerek állnak rendelkezésre abban az esetben, ha egy szimulációs modell szélsőértékét keressük. Ezeket numerikus optimalizálási eljárásoknak nevezzük. Két modellcsaláddal foglalkozunk, a *rács-módszerrel* (*grid search*) és a *gradiens-módszerrel* (*gradient method*).

Rács-módszer. A módszer lényege, hogy a célteret meghatározott számú rácspontra osztjuk és a modellkiértékelést ezekben a pontokban hajtjuk végre. Ezután meghatározzuk az optimumot és ez a pont (illetve ennek paraméterei lesznek) a feladat megoldása(i). Ha feltételezzük, hogy a probléma folytonos és nincs lokális optimuma, – vagy legalább ezek a lokális optimumok kellően messze vannak egymástól, – a rács-módszer sajátmagába ágyazva hatékony módszert ad. Ilyenkor ugyanis megfelelően kiválasztott rácspontok között újra alkalmazni lehet a rács-módszert, természetesen finomítva a rácspontok közötti távolságot.

Nézzünk egy példát! Tegyük fel, hogy a síkban elhelyezkedő n számú pont közé kell elhelyezni egy pontot úgy, hogy az egyes pontoktól mért távolságainak összege minimális legyen. Azaz:

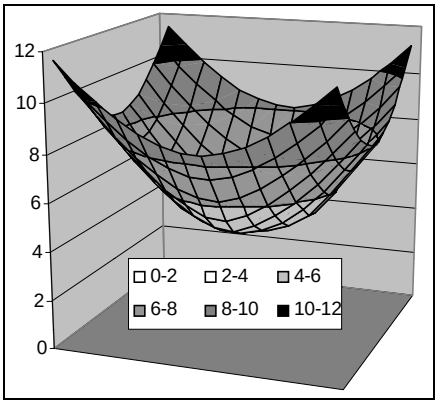
$$\min_{x,y} L\left(x, y, \{a_i\}_1^n, \{b_i\}_1^n\right), \quad L = \sum_{i=1}^{n} \sqrt{(a_i - x)^2 + (b_i - y)^2}\ , \tag{3.1}$$

ahol (a_i, b_i) jelöli az i-edik számú pont koordinátáit, (x, y) pedig a keresett ponttét. (A (3.1) feladat könnyen általánosítható m dimenzióba és K számú optimális pontba, ami nem más, mint a szegmentációhoz alkalmazott *K-közép módszer – K-Mean method*.) A (3.1) feladatnak nincs analitikusan meghatározható megoldása. Ugyanis L deriválásával a következőket kapjuk:

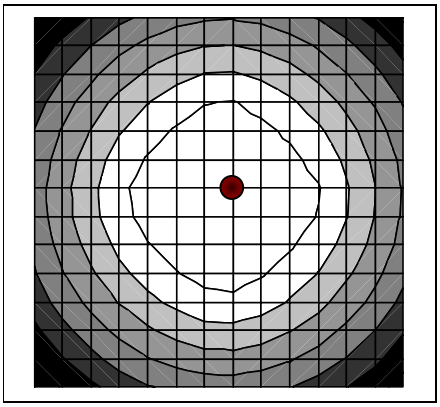
[1] Nincs olyan numerikus módszer, amely pontosan meg tudná határozni $f(x) = (x - \pi)^2$ minimumát.

$$\frac{\partial L}{\partial x} = \sum_{i=1}^{n} \frac{x - a_i}{\sqrt{(a_i - x)^2 + (b_i - y)^2}} = 0$$
$$\frac{\partial L}{\partial y} = \sum_{i=1}^{n} \frac{y - b_i}{\sqrt{(a_i - x)^2 + (b_i - y)^2}} = 0 \qquad (3.2)$$

A (3.2) egyenletrendszerből nem lehet ugyanis zárt alakban meghatározni x-et és y-t. Vessük be a rács-módszert n = 4-re, és a (0,1), (0,-1), (1,0) és (-1,0) pontokra.



3.2. ábra: Rács-módszer (3D)


3.3. ábra: Rács-módszer (2D)

A megoldáshoz a (-2.0; -1.7; -1.4; …; 1.9) × (-2.0; -1.7; -1.4; …; 1.9) rácspontokat használtuk. A 3.2. és 3.3. ábrán a kiértékelt rácspontokat össze is kötöttük és így meghatározhattuk a szintvonalakat. A 3.3. ábrán kövér ponttal jelöltük a rács-módszer által megtalált optimum-pontot, a (0.1, 0.1)-et, ahol L = 4.02. A megoldást visszahelyettesítve (3.2)-be látható, hogy még nem értük el a minimumot. Könnyű belátni azt is, hogy a (3.1) feladatnak csak egy minimumhelye van, továbbá a feladat folytonos (x, y)-ban, ezért megtehetjük, hogy a legjobb rácspont körüli rácspontokból újraindítjuk a keresést finomabb beosztással, például (-0.2; -0.1; …; 0.4) × (-0.2; -0.1; …; 0.4). Ha azonban azt feltételezzük, hogy nem ismerjük a feladat analitikus felírását, csupán egy *fekete-doboz (black-box*) függvényt értékelünk ki, a rácspontok vizsgálata – és vizualizációja – alapján, akkor is élhetnénk a finomítási lehetőséggel.

A rács-módszerről további hasznos információk és tételek találhatók Quandt [1983] és Fletcher [1980] művekben.

Gradiens módszerek. A gradiens módszerek *iteratív kereséssel* próbálják megtalálni az optimális megoldást. Az iteratív keresés lényege, hogy kiindulunk egy lehetséges értékből, majd ezt úgy módosítjuk, hogy lehetőleg minél közelebb kerüljünk a megoldáshoz.

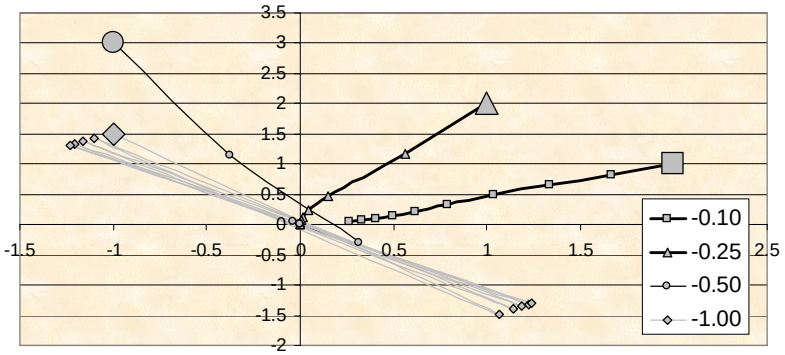
Formálisan:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \lambda_t \boldsymbol{\Delta}_t \,, \tag{3.3}$$

ahol λ jelenti a lépésközt, $\boldsymbol{\Delta}$ pedig az elmozdulás irányát. Gradiens-módszerek alkalmazásánál feltesszük, hogy a célfüggvény deriválható – sőt, bizonyos esetekben kétszer is deriválható. Ebben az esetben akkor juthatunk közelebb egy szélsőérték-helyhez, ha a gradiens irányában mozdulunk el. A (3.3) esetén ez azt jelenti, hogy $\boldsymbol{\Delta}_t = \mathbf{H}_t \, \mathbf{g}_t$, ahol $\mathbf{H}_t$ egy megfelelő pozitív definit mátrix, $\mathbf{g}_t$ pedig az L célfüggvény gradiense, azaz $\mathbf{g}_t = \mathbf{g}(\boldsymbol{\theta}_t) = L'(\boldsymbol{\theta}_t)$. Az egyes gradiens-módszerek általában λ_t és $\mathbf{H}_t$ megválasztásában különböznek. A legegyszerűbb eset, ha $\mathbf{H}_t$-t az egységmátrixnak, **I**-nek választjuk. Ezt nevezzük a *legmeredekebb irány módszerének*. E módszer alkalmazásánál sem mindegy, hogy milyen λ_t paramétert választunk. Nézzük előző, (3.1)-es példánkat. A legmeredekebb irány módszere a következőt adja:

$$\begin{aligned} x_{t+1} &= x_t + \lambda_t \sum_{i=1}^{n} \frac{x_t - a_i}{\sqrt{(a_i - x_t)^2 + (b_i - y_t)^2}} \\ y_{t+1} &= y_t + \lambda_t \sum_{i=1}^{n} \frac{y_t - b_i}{\sqrt{(a_i - x_t)^2 + (b_i - y_t)^2}} \end{aligned} \tag{3.4}$$

Mivel minimalizálást hajtunk végre, ezért λ_t-t negatívnak kell megválasztanunk. A 3.4. ábrán jól látszik, hogy amennyiben a lépésközt – abszolút értékben – kicsinek választjuk (λ_t = -0.1), abban az esetben lassabban konvergálunk az optimális megoldáshoz. Ha azonban túl nagynak választjuk (λ_t = -1.0), abban az esetben ahelyett, hogy közelednénk, távolodunk az optimális megoldástól.



3.4. ábra: Legmeredekebb irány módszer

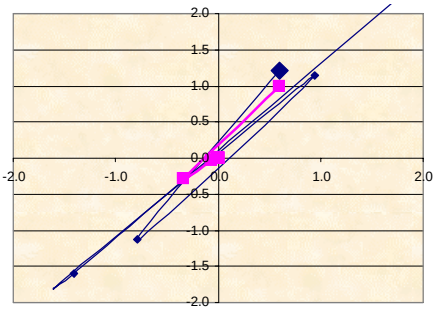
A szélsőérték közelében meg lehet határozni azt a lépésközt, amely a leggyorsabb konvergenciát eredményezi:

$$\lambda_t = \frac{-\mathbf{g}_t'\mathbf{g}_t}{\mathbf{g}_t'\mathbf{G}_t\mathbf{g}_t}, \qquad \mathbf{G}_t = \frac{\partial^2 L(\boldsymbol{\theta}_t)}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}'} \tag{3.5}$$

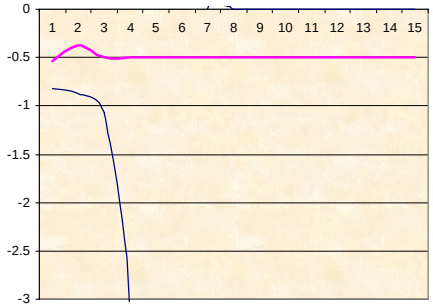
Ha kiszámítjuk a megfelelő értékeket a (3.1) példán, akkor a következőket kapjuk:

$$\begin{aligned}
&\mathbf{g}_t = \begin{bmatrix} \sum_i \frac{a_t - x_i}{\sqrt{T_{i,t}}} \\ \sum_i \frac{b_t - y_i}{\sqrt{T_{i,t}}} \end{bmatrix}, \\
&\mathbf{G}_t = \begin{bmatrix} \sum_i (y_i - b_t)^2 T_{i,t}^{-3/2} & -\sum_i (x_i - a_t)(y_i - b_t) T_{i,t}^{-3/2} \\ -\sum_i (x_i - a_t)(y_i - b_t) T_{i,t}^{-3/2} & \sum_i (x_i - a_t)^2 T_{i,t}^{-3/2} \end{bmatrix}, \\
&T_{i,t} = (x_i - a_t)^2 + (y_i - b_t)^2, \\
&\lambda_t = \frac{-\left(\left(\sum_i \frac{a_t - x_i}{\sqrt{T_{i,t}}}\right)^2 + \left(\sum_i \frac{b_t - y_i}{\sqrt{T_{i,t}}}\right)^2\right)}{S}, \\
&S = \left(\sum_i (y_i - b_t)^2 T_{i,t}^{-3/2}\right)\left(\sum_i \frac{a_t - x_i}{\sqrt{T_{i,t}}}\right)^2 - \\
&-2\sum_i (x_i - a_t)(y_i - b_t) T_{i,t}^{-3/2}\left(\sum_i \frac{a_t - x_i}{\sqrt{T_{i,t}}}\right)\left(\sum_i \frac{b_t - y_i}{\sqrt{T_{i,t}}}\right) + \\
&+\left(\sum_i (x_i - a_t)^2 T_{i,t}^{-3/2}\right)\left(\sum_i \frac{b_t - y_i}{\sqrt{T_{i,t}}}\right)^2.
\end{aligned} \tag{3.6}$$

A probléma azonnal látszik a (3.6) levezetésből, a második deriváltak meghatározása ugyanis igen bonyolult tud lenni. Sőt, abban az esetben, ha $\mathbf{G}_t$ negatív definitté válik, – ami elképzelhető abban az esetben, ha $\boldsymbol{\theta}_t$ messze kerül az optimumtól, – akkor egyre távolabb fogunk kerülni a megoldástól. A 3.5.–3.6. ábrán két induló értékből indítottuk az optimális λ_t lépésközzel számított legmeredekebb irány módszert. Ha a minimumhelyhez elég közeli indulóértéket tudunk választani (0.6; 1), akkor a módszer igen gyorsan konvergál (vastag vonal). Ha azonban egy kicsit messzebbről indulunk (0.6; 1.205), akkor menthetetlenül divergálni fog a sorozat (vékony vonal). Látható, hogy a megfelelő lépésköz megválasztása kritikus fontosságú.



3.5. ábra: Konvergáló és divergáló értékek



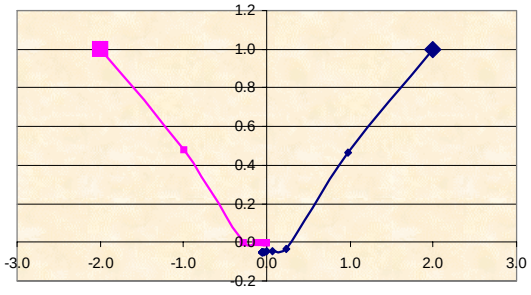
3.6. ábra: Az optimális λ értékek

Megjegyezzük, hogy megfelelő λ_t választással a legmeredekebb irány módszere általában 10-15 kiértékelést igényelt az igen pontos (10^{-8}) optimum megtalálásához. A rács-módszer alkalmazásánál 14^2 kiértékelés után is csak közelítőleg tudtuk az optimumot. Ne felejtsük azonban el, hogy a (3.4) alkalmazásánál feltettük, hogy ismerjük (3.1) analitikus formáját, ami ráadásul folytonosan deriválható.

Ha nem ismerjük a célfüggvényt, illetve a derivált nem hozható zárt alakra, akkor is alkalmazható legmeredekebb irány. Ebben az esetben a deriváltat numerikus módszerrel kell meghatározni, pl.:

$$f'(x) \approx \frac{f(x+h_t)-f(x)}{h_t}, \qquad f''(x) \approx \frac{f(x+2h_t)-2f(x+h_t)+f(x)}{h_t^2}. \quad (3.7)$$

Természetesen ügyelni kell h_t megválasztására, általában $h_t < \lambda_t$. Nézzük, hogyan konvergál a legmeredekebb irány módszer a (3.1) feladat esetében, ha azt feltételezzük, hogy nem ismerjük az analitikus formában problémát. Két különböző h_t érték esetét látjuk a 3.7. ábrán. A megoldás pontossága most ettől a h_t paramétertől is függ.



3.7. ábra: Konvergencia becsült derivált esetén

A legmeredekebb irány algoritmussal az egyik legnagyobb probléma, hogy sok esetben rendkívül lassan konvergál. A másik gradiens-módszer, amelyet bemutatunk, bizonyos feladatok esetén hatékonyabban működik. Ezt a módszert *Newton-módszernek* hívják. Az eljárás abból indul ki, hogy a szélsőérték-helyen a függvény első deriváltja zérus. Ha a deriváltat az elsőrendű Taylor-polinommal közelítjük, akkor a következőt kapjuk:

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbf{q} + \mathbf{G}\left(\boldsymbol{\theta} - \boldsymbol{\theta}^0\right) = 0\,, \tag{3.8}$$

ahol $\mathbf{q}$ és $\mathbf{G}$ rendre L $\boldsymbol{\theta}^0$ pontban vett gradiens vektorát és másodrendű vegyes parciális derivált mátrixát jelöli. Az egyenletet megoldva a következő formához jutunk:

$$\boldsymbol{\theta} = \boldsymbol{\theta}^0 - \mathbf{G}^{-1}\mathbf{q}\,, \tag{3.9}$$

ahonnan már csak egy lépés a (3.3) általánosítása. Így a keresett iterációs forma a következő:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{G}_t^{-1}\mathbf{q}_t\,, \tag{3.10}$$

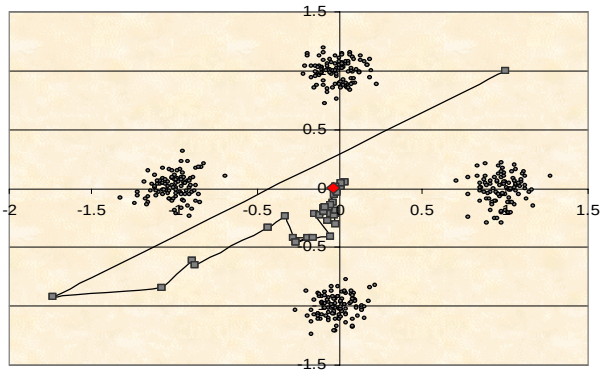
azaz $\mathbf{H} = -\mathbf{G}^{-1}$ és $\lambda = 1$. A legtöbb probléma esetén (3.10) gyorsan konvergál, de természetesen minden olyan fenntartást, amelyet a legmeredekebb iránnyal szemben említettünk, itt is szem előtt kell tartanunk.

Szimuláció optimalizálása esetén még egy problémával szembesülünk. Nem elég, hogy *fekete-doboz* függvény szélsőérték-helyét kell megkeresnünk, de ez még ráadásul sztochasztikusan is viselkedik. A legmeredekebb irány módszert itt is megpróbálhatjuk alkalmazni. Ebben az esetben azonban még jobban kell ügyelni λ_t és h_t megválasztására. Tegyük fel a következőket:

$$\sum_{t=1}^{\infty} \lambda_t = \infty, \qquad \sum_{t=1}^{\infty} \left(\frac{\lambda_t}{h_t}\right)^2 < \infty. \tag{3.11}$$

Ebben az esetben – bizonyos feltételek mellett[2] – bizonyítani lehet, hogy a *sztochasztikus approximációval* ellátott legmeredekebb irány módszer konvergálni fog az optimális megoldáshoz. Ezt az eljárást *Kiefer-Wolfowitz-eljárásnak* nevezi a szakirodalom.

[2] A feltételek között szerepel, hogy a véletlen tényezők függetlenek legyenek, továbbá az optimalizálni kívánt eredeti függvény bizonyos folytonossági tulajdonságoknak eleget tegyen. A feltételeket és a részletes bizonyítást ld.: Kátai [1981].



3.8. ábra: Konvergencia becsült derivált esetén

A 3.8. ábrán bemutatjuk, hogyan alkalmaztuk az eljárást a (3.1) feladat módosított változatára. A módosítás a következő alakú:

$$L = \sum_{i=1}^{n} \sqrt{\left(a_i + 0.1\varepsilon_i - x\right)^2 + \left(b_i + 0.1\varepsilon_i' - y\right)^2}, \qquad \varepsilon_i, \varepsilon_i' \sim N(0,1) \qquad (3.12)$$

Minden egyes szimuláció futtatásakor más-más végeredményt kapunk, azaz úgy értelmezhetjük a problémát, hogy a megfigyelő nem tudja pontosan megmérni a távolságot az egyes pontok között. A Kiefer-Wolkowitz-eljárás alkalmazásakor h_t értékét 0.1-nek (konstansnak) választottuk, míg a lépésközt a $\lambda_t = {}^1/_{3t}$ képlet adta. Így teljesül a (3.11) feltétel és a 3.8. ábrán látható módon sikerült az optimális megoldásba konvergáltatni a sorozatot. Megjegyezzük azt is, hogy mivel az eljárás során a deriváltakat is becsültük, ezért egy lépés megtételéhez összesen háromszor kellett a szimulációs modellt kiértékelni. Először ki kell értékelni az indulópontot, majd az x változó szerinti deriváláshoz egy x-ben módosított, az y szerintihez meg egy y-ban módosított függvényt, azaz:

$$\begin{aligned}
\Delta x_t &= \frac{L(x_t + h_t, y_t) - L\left(x_t, y_t\right)}{h_t} \\
\Delta y_t &= \frac{L(x_t, y_t + h_t) - L\left(x_t, y_t\right)}{h_t} \\
x_{t+1} &= x_t + \lambda_t \Delta x_t \\
y_{t+1} &= y_t + \lambda_t \Delta y_t
\end{aligned} \qquad (3.13)$$

3.3. A genetikus algoritmus

Most, hogy megismerkedtünk a numerikus optimalizálás nehézségeivel, térjünk rá a *genetikus algoritmus* tárgyalására. A genetikus algoritmust 1975-ben fedezte fel Holland [1975]. Azóta az algoritmus számos továbbfejlesztést, módosítást megélt, és különböző változatait alkalmazzák. Az alapalgoritmust Pham-Karaboga [1998] műve alapján ismertetjük.

A genetikus algoritmus hátterében a "*Séma-elmélet*" húzódik. A *séma* az egyes egyedek valamilyen halmazát reprezentálja, azaz a populáció valamely részhalmazát jelenti. Séma lehet például a négyjegyű számok halmazán a 4*2* reprezentáció, ahol a * helyére tetszőleges értékek kerülhetnek. Egy sémát két paraméter határoz meg; a hossza és a rendje. A hosszúság az első és utolsó fix számjegy között található számjegyek számát, a rend pedig a meghatározott értékű számjegyek számát jelenti. Az elmélet alapján tehát az egyes sémák eloszlása egyik generációról a másikra mindig a séma rendjétől, hosszától és túlélési értékétől függ. A genetikus algoritmus célja, hogy meghatározza azt a sémát (génkombinációt), amely az adott probléma szempontjából optimális.

A genetikus algoritmusok gyakorlatilag semmit sem tudnak (semmit sem használnak ki) az optimum problémával kapcsolatosan, és nem foglalkoznak közvetlenül a paraméterekkel. Ehelyett kódokkal dolgoznak, melyek az egyes paramétereket reprezentálják. Ezért az első kérdés az, hogy miképp kódoljuk a problémát, hogyan ábrázoljuk a paramétereket. Mivel a genetikus algoritmus a lehetséges megoldások populációjával foglalkozik, és nem egy darab lehetséges megoldással, ezért a második kérdés az, hogy miképpen határozzuk meg a kezdeti populáció egyedeit. A harmadik kérdés az, hogy a további generációk meghatározásához (fejlődés) melyek a megfelelő genetikus műveletek (öröklődés). A negyedik kérdés a visszacsatolás kérdése, azaz a célfüggvény alapján meg kell határozni az egyedek túlélési valószínűségét. Végül az algoritmusnak valamilyen kritériumot kell szabni a keresés befejezésére.

Reprezentáció: Tegyük fel, hogy az $f(\mathbf{x})$ függvény maximumát szeretnénk meghatározni. Ebben az esetben a populáció egyedeit $\mathbf{x}$ különböző értékei képviselik. Kérdés, hogy a számítógép számára milyen formában adjuk meg ezt az értéket. A reprezentáció formájától függően ugyanis igen különböző lehet a genetikus algoritmus eredménye, azaz más-más reprezentációja ugyanannak a problémának különböző pontosságot és futási időt adhat. A numerikus optimalizálás esetében általában két reprezentációs módszert szoktak alkalmazni (Michalewicz [1992], Davis [1991]). A gyakrabban alkalmazott módszer a *bináris kódolás*, azaz az $\mathbf{x}$ változót kettes számrendszerben ábrázoljuk. Ennek az az előnye, hogy ilyenkor a legnagyobb a lehetséges sémáknak a száma. A szakirodalomban nagyszámú különböző bináris kódolási módszer lelhető fel. (pl.: *Uniform coding*, *Gray scale coding*). A másik lehetséges módszer egyszerűen egész

vagy valós vektorként kezeli **x**-et.[3] Bináris kódolás esetén fontos feladat annak meghatározása, hogy hány bitet használjunk az optimalizálandó változó(k) ábrázolására.

Kezdeti populáció: Az optimalizálás megkezdése előtt a genetikus algoritmusnak szüksége van néhány kezdeti megoldásra. Az egyik lehetséges módszer az, hogy a számítógép véletlenszám-generátora segítségével előállítunk néhány **x** értéket, s ezek képviselik az első generáció egyedeit. A másik lehetséges módszer, hogy külső információt (is) felhasználva az optimális megoldás környékéről indítjuk az algoritmust. Ebben az esetben nyilvánvalóan valamilyen előzetes tudásunk van a problémáról, s ezt kihasználva várhatóan az első esetnél rövidebb idő alatt jutunk optimális megoldáshoz.[4]

Genetikus műveletek: Három hagyományos művelet használatos: *szelekció*, *keresztezés* és *mutáció*, és egy további – ritkábban alkalmazott – az *invertálás*. A genetikus optimalizálás során e műveleteknek számtalan fajtáját és paraméterezését alkalmazhatjuk, továbbá nem is szükséges ezen műveletek mindegyikét használnunk, mivel minden egyes művelet egymástól függetlenül működik. Az alkalmazott genetikus műveletek mennyisége és minősége az adott probléma illetve a reprezentáció függvénye. A műveleteket bináris kódolást feltételezve mutatjuk be. (Részletesen ld.: Whitely – Hanson [1989], Baker [1985]).

Szelekció: A szelekciónak az a célja, hogy több olyan egyedet állítson elő, amelynek a túlélési esélye magasabb és kevesebb olyat, amelynek a túlélési esélye alacsonyabb. A szelekció lényegesen befolyásolja, hogy milyen rövid idő alatt kezd az algoritmus az optimális megoldáshoz konvergálni. Általában két módszert alkalmaznak: az *arányos* (*proportional*) – más néven "*rulett-kerék*" (*roulette wheel*) – és a *rangsorolt* (*ranked*) szelekciót. Az előbbi módszert úgy képzeljük el, hogy adott egy óriási rulett-kerék, ahol

---

[3] Egy rövid példával szemléltetjük a két reprezentációs módszert. Tegyük fel, hogy a lehetséges megoldások egész számok, és az optimális megoldás valahol $x = 16$ körül található. Binárisan kódolva 16 = [0 1 0 0 0 0], viszont 15 = [0 0 1 1 1 1]. Ha az utóbbi irányból "közelít" az algoritmus, akkor esetleg olyan eredményt ad, hogy a [0 0 1 1 * *] séma helyes. Ezzel szemben ha egyszerű egész számként kezeljük, akkor az 1* (azaz tizenvalahány) séma a helyes irány.

[4] Általában az első módszert alkalmazzák vagy a két módszert kombinálják. Tegyük fel például, hogy az $f(x,y)$ függvényt szeretnénk maximalizálni, azzal a feltétellel, hogy $x = y$. A genetikus algoritmust ha mint fekete doboz szimulációt alkalmazzuk, akkor a feltételekkel nem tud külön mit kezdeni, és ezért azt az optimalizálandó függvénybe kell beépíteni, pl.: $g(x,y) = f(x,y) - a(x - y)^2$, ahol a egy megfelelően nagy pozitív szám. Véletlen indítás esetén igen kicsi a valószínűsége annak, hogy egymás után két azonos számot generálunk, így nagyon sokáig eltarthat, míg az algoritmus egy egyáltalán lehetséges megoldásra rátalál, majd ezek után vagy azonnal leáll, vagy újfent sokáig tart míg arra a sémára "rájön", ami $x = y$-t jelenti. Ha eleve olyan - akár véletlenül meghatározott - pontokból indítjuk az algoritmust, ahol $x = y$, akkor hamarabb jutunk megoldáshoz. E példát csupán szemléltetésül említjük, nyilvánvaló, hogy sokkal egyszerűbb megoldás az, ha a korlátozott probléma helyett a számítógép a nem korlátozott $f(x,x)$ problémát oldja meg.

minden egyed arányosan annyiszor szerepel, amekkora a túlélési esélye. Annyiszor pörgetjük meg a kereket, ahány egyedet a következő generációba szánunk, így a nagyobb túlélési esélyű egyedek nagyobb valószínűséggel – többször – kerülnek a következő generációba. Rangsorolt szelekció esetén egy korábbi generációból egy egyednek legfeljebb egy replikációja kerülhet a következő generációba. Ekkor minden egyedhez egy véletlen számot generálunk úgy, hogy annak várható értéke annál nagyobb legyen, minél nagyobb az egyed túlélési esélye. Végül e szám alapján rangsoroljuk az egyedeket és kiválasztjuk az első *n* darabot a következő generáció számára.

Keresztezés: Ez az a művelet, amely alapjában megkülönbözteti a genetikus algoritmust más optimalizáló módszerektől. A keresztezés során két létező egyed (*szülők*) két új egyedet (*utódok*) hoz létre. A szülőket szelekcióval határozzuk meg. A keresztezésnek jó néhány különböző fajtáját alkalmazzák (pl.: egy ponton történő (*one-point*), két ponton történő (*two-point*), ciklikus (*cycle*) és egyenletes (*uniform*) keresztezés). A keresztezés töréspontjait külön heurisztika szabályozza, így a keresztezés során egyre közelebb kerülünk az ideális sémához. Az alábbi ábrákon az említett négy keresztezésre mutatunk példát:

1. szülő:	[**1 0 0 0 1 0 0 1 1 1 1**]
2. szülő:	[0 1 1 0 1 1 0 0 0 1 1]
1. utód:	[**1 0 0 0 1** 1 0 0 0 1 1]
2. utód:	[0 1 1 0 1 **0 0 1 1 1 1**]

3.9. ábra: Egy-ponton keresztezés

1. szülő:	[**1 0 0 0 1 0 0 1 1 1 1**]
2. szülő:	[0 1 1 0 1 1 0 0 0 1 1]
1. utód:	[**1 0 0** 0 1 1 0 **1 1 1 1**]
2. utód:	[0 1 1 **0 1 0 0** 0 0 1 1]

3.10. ábra: Két-ponton keresztezés

1. szülő:	[**1 0 0 0 1 0 0 1 1 1 1**]
2. szülő:	[0 1 1 0 1 1 0 0 0 1 1]
1. utód:	[**1** 1 **0** 0 **1** 1 **0** 0 **1** 1 **1**]
2. utód:	[0 **0** 1 **0** 1 **0** 0 **1** 0 **1** 1]

3.11. ábra: Ciklikus keresztezés

1. szülő:	[**1 0 0 0 1 0 0 1 1 1 1**]
2. szülő:	[0 1 1 0 1 1 0 0 0 1 1]
Minta:	[1 1 0 0 1 1 1 1 0 0 0]
1. utód:	[0 1 **0 0** 1 1 0 0 **1 1 1**]
2. utód:	[**1 0** 1 0 **1 0 0 1** 0 1 1]

3.12. ábra: Egyenletes keresztezés

Mutáció: A mutáció során az egyedeket bitről bitre megvizsgálja az algoritmus és véletlenszerűen megváltoztathatja. A mutáció heurisztikája azt jelenti, hogy az egyes bitek milyen eséllyel változzanak meg (*mutációs ráta*), és csakúgy mint a keresztezésnél, ez az érték dinamikusan változhat, annak érdekében, hogy az ideális séma kialakuljon. Szemben a keresztezéssel, a mutációhoz elegendő egy szülő, és az eredmény egy utód, ahol a szülő kiválasztása ismét a szelekcióra van bízva. A mutáció segítségével az algoritmus új területeket fedez fel, ami azért hasznos, mert elvezeti az algoritmust a lokális optimális pontokból a globális optimum felé.

Szülő:	[0 1 0 0 1 **1** 0 0 1 1 1]
Utód:	[0 1 0 0 1 **0** 0 0 1 1 1]

3.13. ábra: Mutáció

Invertálás: A művelet során a szülő-egyed megfelelő heurisztikával kiválasztott bitjei felcserélődnek, azaz 0-ról 1-re, 1-ről 0-ra változnak. Gyakorlatilag a mutáció is felfogható egy egy-bites invertálás műveletnek.

Szülő:	[0 1 0 0 **1 1 0 0** 1 1 1]
Utód:	[0 1 0 0 **0 0 1 1** 1 1 1]

3.14. ábra: Invertálás

Többször is hangsúlyoztuk, hogy az egyes műveleteknek különböző paraméter-beállításai léteznek. Ilyen paraméterek: a *populáció mérete*, a *keresztezési ráta*, illetve a *mutációs ráta*. Számtalan publikáció foglalkozik ezen paramétereknek a genetikus algoritmus sebességére és pontosságára való hatásának vizsgálatával (Schaffer *et al*. [1989], Grefenstette [1986], Fogarty [1989]). Természetesen maga az optimum-probléma képviseli a legjelentősebb hatást a genetikus algoritmus számára, és két különböző probléma esetén ugyanaz a beállítás egészen különböző eredményeket produkálhat. Ennek ellenére megfogalmazható néhány általános megállapítás. Nagy populáció (minta-méret) esetén a számítás lassabb lesz, azonban mivel az egyedek jobban lefedik a lehetséges megoldások halmazát, ezért nagyobb a valószínűsége annak,

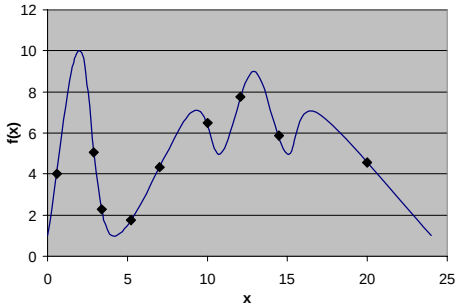
hogy globális optimumot kapjunk. A keresztezési ráta (*crossover rate*) meghatározza, hogy milyen gyakran történjenek a keresztezések. Túl alacsony ráta esetén nagyon lassúvá válhat a konvergencia, míg túl magas ráta esetén elképzelhető, hogy egyetlen megoldás körül ugrálunk. Végül a túlságosan magas mutációs ráta erőteljesen különbözővé teszi a populációt minden egyes lépésben, ezáltal instabilitást okozhat. Másik oldalról a globális optimum megtalálása nagyon nehéz olyan esetben, ha a mutációs ráta túl alacsony.

A genetikus algoritmusokat az teszi nehézzé, hogy a fenti paraméterek megadására nincs egyetlen út. Néhány problématípusról be lehet bizonyítani, hogy adott paraméter-beállítás esetén milyen gyorsan konvergál, illetve mekkora valószínűséggel találja meg az optimális megoldást, sok esetben azonban ismeretlen (fekete doboz) a célfüggvény. Ebben az esetben gépünk kapacitásától függően különböző paraméter-beállításokkal kell próbálkoznunk. Mikor végül elfogadjuk a genetikus algoritmus által optimálisnak vélt megoldást, tisztában kell lennünk azzal, hogy nem 100%-ig biztos, hogy a valódi optimumban vagyunk. Azt azonban állíthatjuk, hogy az adott körülményekhez képest (vagyis, hogy semmit sem tudunk a problémáról), adott idő alatt (számítás idő - kapacitás) a lehető legjobb eredményt kaptuk. Abban az esetben, ha mindenképpen meg akarunk győződni az eredmény helyességéről, vagy előzetes információnk van a célfüggvény tulajdonságairól, más módszerekkel érdemes kombinálni a genetikus algoritmust. (Pl.: *simulated annealing*).

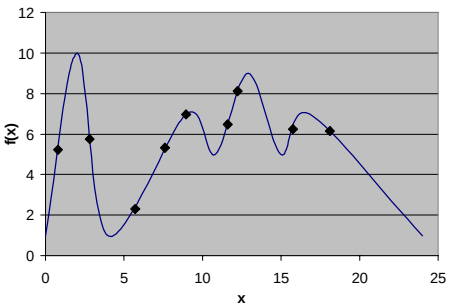
Túlélési valószínűségek: A túlélési valószínűségek meghatározására is igen sok módszert lehet alkalmazni. Abban az esetben, ha a feladatban a célfüggvény mellett egyéb feltételek is szerepelnek, akkor a feltételeket a célfüggvénybe kell integrálni (ld.: 5. lábjegyzet). Nyilvánvaló, hogy maximumfeladat esetén, minél nagyobb az adott egyednek megfelelő pontban a függvény értéke, annál nagyobb az adott egyed túlélési esélye; minimum feladatnál pedig fordítva. A kérdés az, hogy mennyivel. Tegyük fel, hogy 4 egyed túlélési valószínűségét szeretnénk megadni. Legyen a függvényértékük rendre 1, 2, 3 és 4, továbbá legyen a feladat maximum probléma. Az egyik legegyszerűbb számítási lehetőség a *diszkrét korlát* állítása. Legyen ennek a diszkrét korlátnak az értéke mondjuk 2.5, és a korlát alatti egyedek túlélési valószínűsége 0, a korlát felettieké 1. Így az 1. és 2. egyed túlélési valószínűsége 0, a 3. és 4.-é pedig 1. Folytassuk e példát a már ismertetett szelekcióval. Tegyük fel, hogy 3 egyedet kell kiválasztani. Rulett módszer esetén a rulett-keréken 1 darab 3-as és 1 darab 4-es szerepel. 3-szor forgatunk, így átlagosan a következő mintába 1.5 darab 3-as és 1.5 darab 4-es kerül. Elképzelhető az is, hogy adott esetben 3 darab 3-as kerül be, viszont 1-es és 2-es soha. Rangsorolás esetén minden egyedhez rendelnünk kell egy véletlen számot. Legyen ez a véletlen szám a [0; *túlélési valószínűség*] intervallumon egyenletes eloszlású. Adott esetben például rendre a következő értékeket kaphatjuk: 0, 0, 0.89, 0.41. A rangsor pedig emiatt a következő: 3, 4, 1, 2. Három egyedet kiválasztva

megállapíthatjuk, hogy jelen esetben a rangsorolt módszer bármilyen véletlenszámokat is generálunk, mindig a 3, 4 és 1 egyedeket választja ki. Nézzünk még egy módszert a túlélési valószínűség meghatározására, az *egyenletes eloszlást*! Ebben az esetben a függvényértékkel arányos túlélési valószínűségeket generálunk az egyedeknek, azaz a túlélési valószínűségek rendre a következők: 0.1, 0.2, 0.3, 0.4. Bár különböző alkalmazásokban más-más eljárásokat alkalmazhatnak, felhasználhatják korábbi generációk össztúlélési valószínűségeit, más eloszlásokból vehetnek véletlen számokat, stb., a leggyakrabban alkalmazott módszer mégis az említett két eljárás.

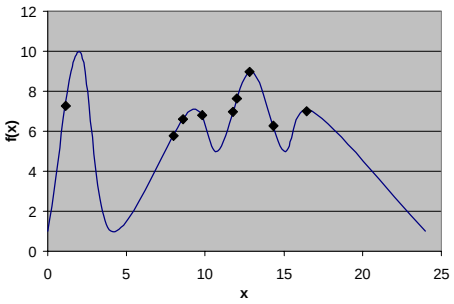
Leállási kritérium: Az algoritmusnak elvileg akkor kellene megállnia, amikor elérte a globális optimumot, gyakorlatilag azonban más helyen is megállhat. Megállhat a globális optimum környékén, megállhat lokális optimumban, annak a környékén, vagy akár nem-optimális pontban. A számítógép csak "sejteni" tudja, hogy hol tart éppen, azt vizsgálva, hogy az újonnan generált egyedek milyen közel vannak az előzőkhöz, és milyen nagy az eltérés a függvényértékekben. Ezt nevezzük *konvergencia-kritériumnak*. Amennyiben az algoritmust ez szabályozza, nagyon valószínű, hogy legalább lokális optimumot kapunk, továbbá minél kevésbé érzékeny a konvergencia-kritérium, annál valószínűbb, hogy a globálisan optimális pont(ok)hoz közel leszünk. Ez azonban megnöveli a számítási időt. Annak érdekében, hogy ne lokális, hanem globális optimumban álljunk meg, egyrészt elegendően nagy számú populációval kell dolgoznunk, másrészt nem szabad sürgetnünk a konvergenciát. A szakirodalom *korai konvergenciának*, illetve *genetikus elsodródásnak* nevezi azt a jelenséget, amikor a globális megoldás helyett az algoritmus lokális optimumban áll le. Egyéb leállási kritériumok is elképzelhetők és használatosak; például meghatározott *futási idő*, vagy meghatározott *ciklus idő* (meghatározott generáció szám). E két utóbbi esetben elvileg még a lokális optimum sem biztosított, illetve elképzelhető, hogy a túl hosszúra választott futási idő miatt az algoritmus már nem mozdul a kialakult stabil állapotból egy idő után. Legtöbbször a konvergencia-kritériumot kombinálják ez utóbbi két kritériummal.



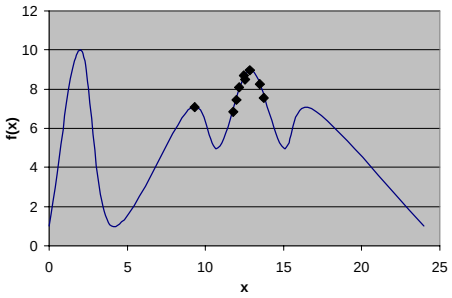
3.15. ábra: Elsodródás (1. generáció)



3.16. ábra: Elsodródás (2. generáció)



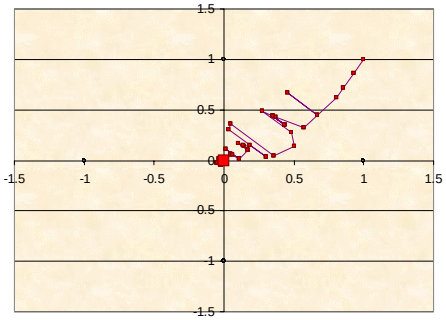
3.17. ábra: Elsodródás (3. generáció)



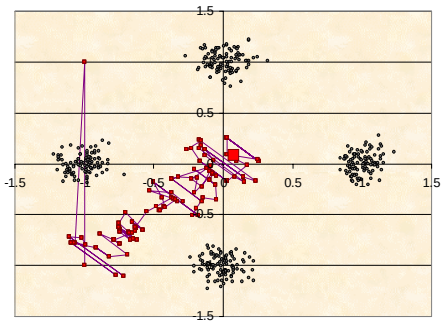
3.18. ábra: Elsodródás (4. generáció)

A 3.15.-3.18. ábrákon vonallal a teljes leszámlálás eredményét, ponttal a genetikus algoritmus néhány generációjának egyes egyedeit jelöltük. Jól látható, hogy az algoritmus fokozatosan eltolódik a lokális optimum felé, és nem a globális optimumban áll le.

A (3.1) probléma és a probléma (3.12) sztochasztikus kiterjesztését egy egyszerű genetikus algoritmus segítségével oldottuk meg. Egy generációban négy egyed versenyzett és a két legjobb egyed kapott lehetőséget szaporodni. A szaporodás egyszerűen abból állt, hogy az egyik szülő *a* koordinátáját kicseréltük a másik szülő *b* koordinátájával. Így két új egyedet kaptunk. A másik két új egyedet mutációval hoztuk létre, ahol a kiválasztott szülők mindkét koordinátáját egy kis mértékben – véletlenszerűen – megváltoztattuk. A 3.19-3.20 ábrákon minden generáció legjobb elemét ábrázoltuk. Numerikus kísérleteink azt igazolták, hogy az alkalmazott genetikus eljárás hasonlóan jó eredményeket ad, mint a megfelelő indulóértékből, megfelelő lépésközzel indított gradiens alapú keresés.



3.19. ábra: Determinisztikus feladat



3.20. ábra: Sztochasztikus feladat

3.4. A neurális háló[5]

A neurális háló két objektumból áll; *csomópontokból* (*neuronok*) és az őket összekötő *élekből* (*szinapszis*). A bejövő információ (inger, impulzus) egy-egy neuronba érkezik. Amennyiben az adott inger egy *küszöbértéket* meghalad, a neuron továbbküld egy jelet abba a neuronba, amellyel összeköttetésben van. Egy ilyen neuronba több neuron is küldhet impulzust. Ha ezek összessége meghaladja e neuron ingerküszöbét, akkor ez a neuron is továbbküld egy impulzust a következő neuronba, stb. Végül az impulzus eljut néhány végső neuronba, s attól függően, hogy melyikbe jutott el, úgy kötünk valamilyen válaszreakciót az adott inputhoz. Tanulás során az intenzíven használt szinapszisok megerősödnek, a keveset használtak pedig megszűnnek. Az emberi agy tanulásának ezt a mechanizmusát utánozza le a számítógépes modell. Egyes elemeket azonban meg kellett változtatni, ugyanis még a leggyorsabb számítógépek sem képesek néhányszáz neuronnál nagyobb hálózatot modellezni – szemben az aggyal, amelyben milliárdnyi neuron működik. Az első változtatás az, hogy az egyes neuronok több más neuronnak is adhatnak impulzust, akár visszafelé is. A második változtatás, hogy minden szinapszisnak meghatározott az impulzus-átadási hatásfoka, azaz meghatározott erővel (*súllyal*) továbbítja az információt. A harmadik változtatás az, hogy az ingereket fogadó neuronokat különböző mértékű (intenzitású) ingerek érhetik, továbbá az ingerek továbbítására nem egy egyszerű küszöbérték-meghaladási kritérium létezik, hanem bonyolult függvények segítenek meghatározni a kimenő impulzus mértékét. Kivétel általában a beérkező ingereket fogadó neuron, ahonnan is az inger "egy-az-egyben" továbbítódik. Végezetül, a tanulás mechanizmusát matematikai-statisztikai módszerek szabályozzák, melyek közül a legismertebb és leginkább használatos módszer a *hiba-visszaterjesztési* (*back-propagation*) eljárás.

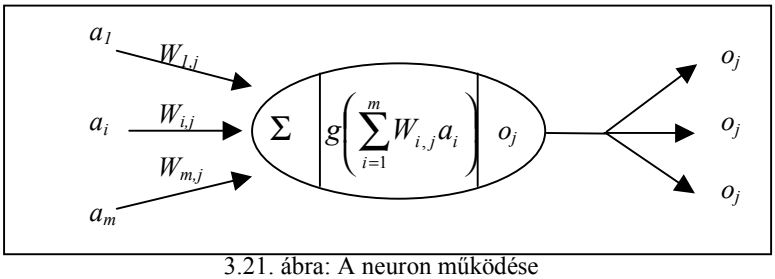
Vizsgáljuk meg először a neurális háló alapegységét, a neuront. Ezt az egységet gyakran nevezik *számítási* (vagy *processzáló*) elemnek is, mivel itt zajlanak le a nemlineáris számítási folyamatok. A neuron gyakorlatilag nem más, mint egy függvény, ami a bemeneti jeleket egyetlen kimeneti jellé alakítja. E függvényeknek általában meghatározott alakjuk van. Először a bemeneti jeleket összegzi valamilyen módon (*additív*, *multiplikatív*), majd egy függvény segítségével transzformálja ezt a jelet. A leggyakrabban alkalmazott függvények a (3.14) ugrásfüggvény és a (3.15) szigmoid függvény:

$$U(x,t)=\begin{cases}1, & \text{ha } x \geq t\\ 0, & \text{ha } x < t\end{cases} \tag{3.14}$$

[5] Elsősorban Fu [1994], Eaton, Griffith [2000] és Rumelhart et al. [1986] munkáira építünk. Magyar nyelven ld. Russel, Norvig [2000] nemrég lefordított könyvét.

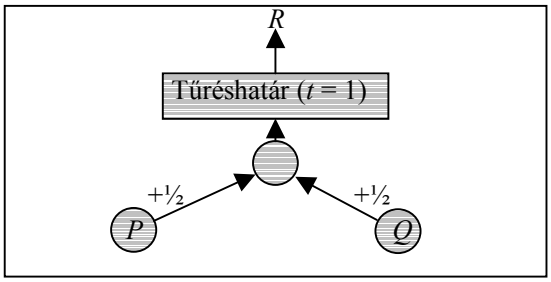
$$S(x) = \frac{1}{1+e^{-x}}. \tag{3.15}$$

Az összegzésre általában additív módszert alkalmaznak. A neuronokat összekötő éleken súlyok szerepelnek, amelyek az adott jeleket felerősíthetik vagy gyengíthetik. Ha tehát egy *j* neuronba *m* számú más neurontól érkezik $W_{i,j}$ súlyokon keresztül a_i jel, akkor ezt a következőképpen ábrázolhatjuk:



3.21. ábra: A neuron működése

Nézzünk egy egyszerű példát a 3.22. ábra alapján! Példánkban két bemeneti neuronunk van. Az ide beérkező jel változtatás nélkül, de természetesen súlyozva kerül be a kimeneti neuronba. Itt megtörténik a jelfeldolgozás, majd a kiküldött érték lesz a neuronháló végeredménye. Számítási függvénynek most az ugrásfüggvényt használtuk, méghozzá $t = 1$ paraméterrel. (A t paramétert *tűréshatárnak* is nevezik). A súlyokat egységesen válasszuk ½-nek.



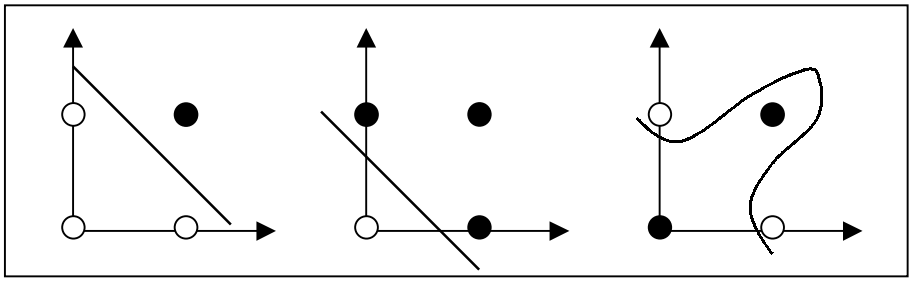
3.22. ábra: Az ÉS függvény neurális reprezentációja

Ha feltesszük, hogy *P* és *Q* csak bináris értékeket vehet fel, akkor a hálózat pontosan a *logikai ÉS függvényt* reprezentálja. Ugyanis:

P	*Q*	Súlyozott összeg	Tűréshatár	Kimenet
0	1	½ · 0 + ½ · 1 = ½	½ < 1	0
1	0	½ · 1 + ½ · 0 = ½	½ < 1	0
1	1	½ · 1 + ½ · 1 = 1	1 = 1	1
0	0	½ · 0 + ½ · 0 = 0	0 < 1	0

3.1. táblázat: Az ÉS függvény reprezentálása

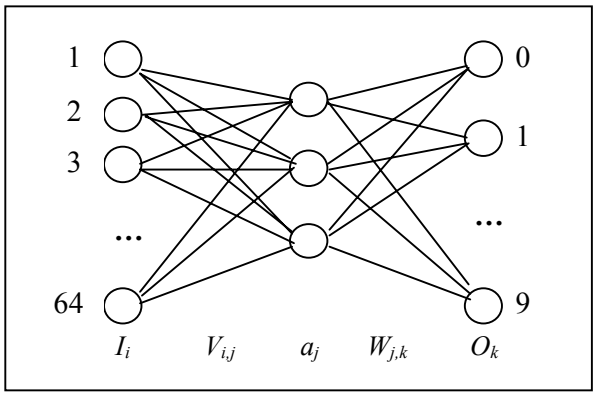
A 3.22. ábrán látható neurális hálózat egy *egyrétegű* hálózati struktúra, ugyanis a bemeneti neuronok közvetlen összeköttetésben vannak a kimeneti neuronokkal. Minsky és Papert [1969, 1988] komolyan foglalkoztak a neurális hálózatok működésének matematikailag egzakt formába való öntésén, és ennek keretében többek között bebizonyították, hogy minden egyrétegű – közbülső réteg nélküli – neurális háló csak olyan függvények reprezentálására képes, amelyek lineárisan szeparálhatók. Valóban, a logikai *kizáró-vagy* függvény (XOR) nem lineárisan szeparálható, és egyetlen réteg segítségével nem lehetséges a függvény neurális megfelelőjét létrehozni (ld.: 3.23. ábra).



3.23. ábra: Az ÉS, a VAGY és a kizáró-VAGY függvény grafikus reprezentációja

Ha többrétegű hálózatot kívánunk létrehozni, akkor a bemeneti és kimeneti rétegek közé egy (vagy több) *köztes* – vagy *rejtett* – *réteget* kell beékelnünk. A 3.24. ábrán egy olyan neurális hálózat struktúrája látható, ahol a bemeneti réteg 64 darab, a köztes réteg 3 darab, a kimeneti réteg pedig 10 darab neuronból áll. Mivel a bemeneti neuronok nem számító egységként vannak a hálózatban, ezért a hálózat egy olyan függvény, amelyben 13 nemlineáris függvény van meghatározott súlyokkal összekapcsolva. Mivel a súlyok első csoportja a 64 bemenetből a 3 köztes neuronba juttatja el a jelet, ezért itt összesen 192 súly szerepel. A 3 köztes neuronból a 10 kimeneti neuronhoz összesen 30 súly kell. Ez azt jelenti, hogy ha például szigmoid függvény segítségével dolgozunk, akkor a 3.24. ábrán egy 222 szabad paraméterrel rendelkező bonyolult nemlineáris függvényről van szó. Ha hozzávesszük még azt is, hogy minden processzáló egységhez biztosítani szoktak egy konstans neuront is, az azt jelenti, hogy a szabad paraméterek száma további 13-mal növekszik. Az ábrán látható neurális háló analitikusan is kifejthető:

$$a_j = g\left(C_j^a, \sum_{i=1}^{64} V_{i,j} I_i\right) = \frac{1}{1+e^{-\sum V_{i,j} I_i + C_j^a}}$$
$$O_k = g\left(C_k^O, \sum_{j=1}^{3} W_{j,k} a_j\right) = \frac{1}{1+e^{-\sum W_{j,k} a_j + C_k^O}} \qquad (3.16)$$



3.24. ábra: Többrétegű neurális háló

A többrétegű neurális hálózatok már nem csak lineárisan szeparálható függvények reprezentációjára képesek. A következő tételt Cybenko [1988, 1989] fogalmazta meg és bizonyította be.

3.1. Tétel: *Ha egy neurális hálózat legalább egy rejtett réteget tartalmaz, akkor tetszőleges folytonos függvény reprezentálására képes. Ha egy hálózat két rejtett réteggel rendelkezik, akkor tetszőleges függvény reprezentációjára képes.*

A neurális hálózatok alkalmazásának kritikus pontja a *tanulás*. Ez az a folyamat, amikor a minták által megadott inputokhoz (I_i) és outputokhoz (O_k) meg akarjuk találni a tökéletes hálózati struktúrát – *topológiát* – és súlyrendszert.[6] A súlyok meghatározása a viszonylagosan egyszerűbb kérdés. Az egyrétegű hálózatokra az első tanulási módszert Hebb [1949] után *Hebb-féle tanulásnak* nevezik. Az algoritmus lényege a következő:

1. A tanuló mintából válasszunk ki véletlenszerűen egy tanuló halmazt (I_i, T_k – párt),
2. Számítsuk ki a megadott input alapján a hálózat outputját (O_k)
3. Hasonlítsuk össze a számított és a tényleges eredményt, és módosítsuk a súlyokat úgy, hogy $W_{i,k}^{t+1} = W_{i,k}^t + \lambda(T_k - O_k)I_i$.

[6] Vegyük észre, hogy itt tulajdonképpen egy nemlineáris regresszió építéséről van szó.

Az algoritmusból jól látszik, hogy csak abban az esetben történik változtatás a súlyokon, ha a számított és tényleges eredmény nem egyezik meg. Próbáljuk most megtanítani egy egyszerű 2 input 1 output neuronos hálózatnak az *ÉS* szabályt. Tegyük fel, hogy kiinduló esetben a súlyok véletlenszerűen voltak kiválasztva és értékük –0.1 és +0.2. A mintákat véletlenszerűen húzzuk és feltételezzük, hogy $\lambda = 1$:

Q	*P*	W_O	W_P	Összeg	Tűréshatár	*R*	Tény	Új W_O	Új W_P
0	1	–0.1	+0.2	+0.2	+0.2 < 1	0	0	–0.1	+0.2
1	0	–0.1	+0.2	–0.1	–0.1 < 1	0	0	–0.1	+0.2
1	1	–0.1	+0.2	+0.1	+0.1 < 1	**0**	**1**	+0.9	+1.2
0	1	+0.9	+1.2	+1.2	+1.2 > 1	**1**	**0**	+0.9	+0.2
1	0	+0.9	+0.2	+0.9	+0.9 < 1	0	0	+0.9	+0.2
1	1	+0.9	+0.2	+1.1	+1.1 > 1	1	1	+0.9	+0.2
0	1	+0.9	+0.2	+0.2	+0.2 < 1	0	0	+0.9	+0.2
0	0	+0.9	+0.2	+0.0	+0.0 < 1	0	0	+0.9	+0.2

3.2. táblázat: Az ÉS függvény tanulása

A többrétegű neuronhálózatok leggyakrabban alkalmazott súlytanulási algoritmusa a *hiba-visszaterjesztés* módszer. A módszer lényege – hasonlóan a Hebb-féle tanuláshoz – az, hogy úgy kell megváltoztatni a súlyokat, hogy az adott minta esetében az elkövetett hiba csökkenjen. Mivel több réteg működik együttesen, ezért a hibát hátulról kell visszaterjeszteni az egyre előbbi súlyokig. Azaz először a rejtett és output rétegek közötti súlyokat kell módosítani:

$$W_{j,k}^{t+1} = W_{j,k}^{t} + \lambda a_j \left(T_k - O_k\right) g'\left(\sum_j W_{j,k} a_j\right), \tag{3.17}$$

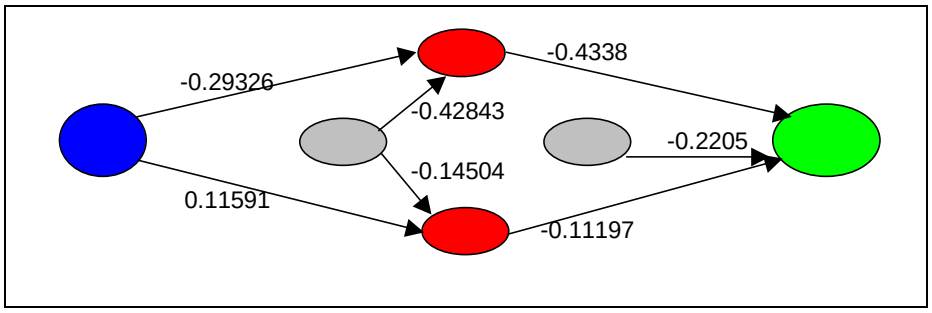
ahol *g*-vel jelöltük a processzáló függvényt, amelyet *aktiváló-* vagy *reakciófüggvény* néven is említi a szakirodalom. Látható, hogy a szigmoidfüggvény alkalmazása azért kedvező, mert könnyen számítható a deriváltja, ugyanis $g' = g(1 - g)$. A köztes és a bemeneti neuronok között szintén valami hasonló szabályt kellene megállapítani a súlyok megváltoztatása érdekében. Itt azonban nem egyszerű az elkövetett hiba mértékének meghatározása, hiszen amíg a kimeneti értéket össze lehetett hasonlítani a tényadattal, addig itt nem áll rendelkezésre a tényadathoz hasonló mennyiség. Az ötlet az, hogy minden *j* rejtett neuron valamilyen mértékben hozzájárul minden *k* kimeneti csomópontban elkövetett hibához. Ezért a már meghatározott kimeneti hibaértékeket a csomópontok közötti kapcsolat szorossága, azaz a súlyok függvényében osztjuk szét, és visszaterjesztjük azokat a rejtett csomópontokhoz. Azaz a *j*-edik rejtett neuron hibatényezője:

$$\Delta_j^t = \sum_k \left(W_{j,k}^t \left(T_k - O_k \right) g' \left(\sum_j W_{j,k}^t a_j \right) \right), \tag{3.18}$$

és így a bemeneti és a köztes neuronok közötti súlyok módosítása már hasonló a (3.17) képlethez:

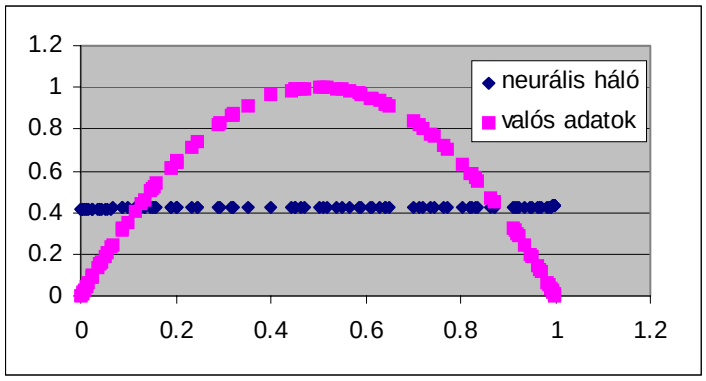
$$V_{i,j}^{t+1} = V_{i,j}^t + \lambda I_i \Delta_j^t g' \left(\sum_i V_{i,j} I_i \right). \tag{3.19}$$

Szemléltetésül a következő példában bemutatunk egy egyszerű neurális modellt. A feladat az, hogy a hálózat megtanulja az $f(x) = 4x(1-x)$ függvényformát az $x \in [0, 1]$ intervallumon. A függvényt természetesen egy mintahalmaz segítségével kell megtalálni, és ehhez véletlenszerűen kiválasztottunk 100 különböző x értéket. A neurális hálózat első indításra a következő formát öltötte:



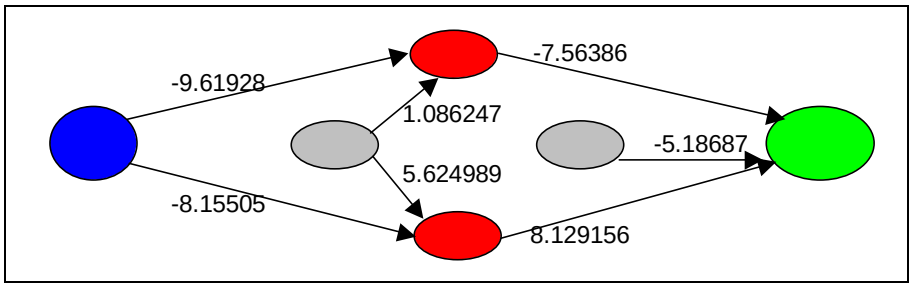
3.25. ábra: Kiinduló hálózat – véletlen súlyokkal

Látjuk, hogy egy nagyon egyszerű topológiát választottunk, hiszen egyetlen bemeneti érték, $I = x$, és egyetlen kimeneti érték $T = f(x)$ van. Köztes neuronból is elég volt kettő. A látható súlyok véletlen számok. Nézzük meg hogyan halad az információ a bemeneti neuronból a kimeneti neuronba. Legyen például x értéke 0.2. Ekkor $f(x) = 4 \cdot 0.2 \cdot 0.8 = 0.64$. Milyen érték kerül a *HID1* neuronba? A megfelelő súllyal szorzott bemeneti érték és a konstans, azaz $-0.29326 \cdot 0.2 - 0.42843 = -0.487082$. Ez az érték a *HID1* neuronban átalakul a reakciófüggvény segítségével. Jelen esetben a szigmoid függvényt alkalmaztuk, így a behelyettesítés után $\frac{1}{1+e^{0.487082}} = 0.380581$. A *HID2* és az *OUTPUT* neuron értéke is hasonlóan határozódik meg. Így végül a modell 0.64 helyett 0.410467-at tippel. Hasonlóan tippeli a maradék 99 x értéket. Mindezek után azt tapasztaljuk, hogy a függvényformát elég rosszul becsüli (nyilvánvalóan a tanulás folyamata előtt, ld.: 3.26. ábra).



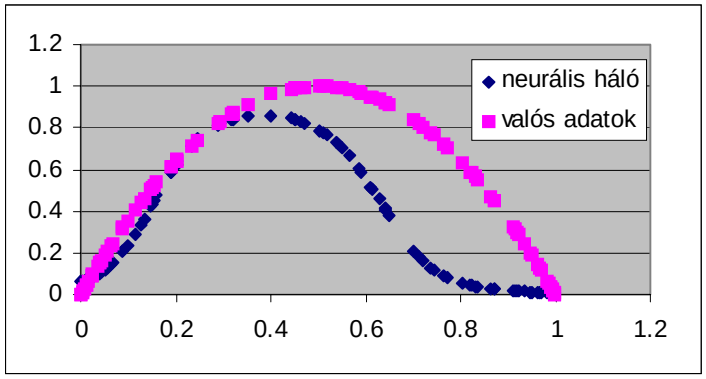
3.26. ábra: A kiinduló hálózat eredményei

A (3.17) és (3.19) képletekkel definiált tanulás során kiragadunk egy nem tökéletes, „félkész” neurális hálót, és megvizsgáljuk, hogyan halad a tanulás, (ld.: 3.27 ábra).



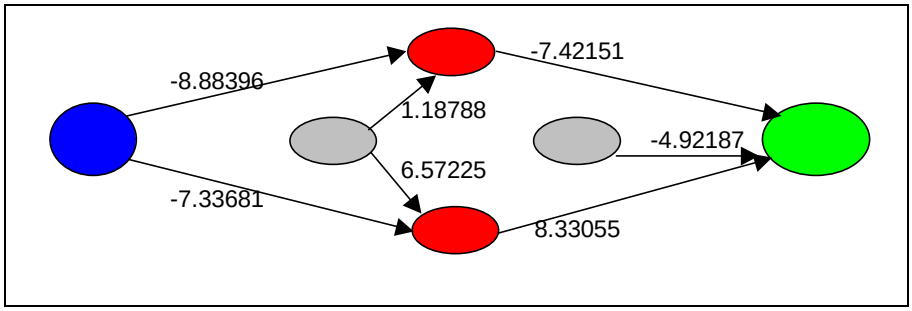
3.27. ábra: Félkész hálózat – száz iteráció után kialakított súlyokkal

A 3.27 ábrán látható súlyokat alkalmazva ismét megvizsgáljuk mennyire sikerült illeszkedni az eredeti pontokhoz, (ld.: 3. 28. ábra).



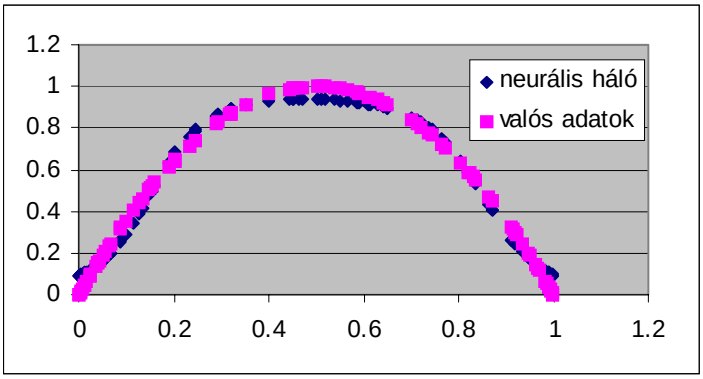
3.28. ábra: A félkész hálózat eredményei

Látjuk, hogy az adatsor egy részét már sikerült jól megtanulni, míg egy másik rész még hátra van. Végül a tanulási folyamat az alábbi modellnél fejeződött be, azaz a neurális hálózatban szereplő súlyok az alábbi értékekhez konvergáltak, (ld.: 3.29 ábra).



3.29. ábra: Végső neurális háló

A 3.29. ábrán látható súlyokat felhasználva a neurális hálózat által becsült értékek jól simulnak a feladatban meghatározott függvény megfelelő értékeihez. A tanulás befejeződött.



3.30. ábra: A végső hálózat eredményei

Nézzünk most egy kicsit a hiba-visszaterjesztési algoritmus matematikája mögé!

3.2. Tétel: *A* (3.17) *és* (3.19) *képleteken alapuló hiba-visszaterjesztési eljárás nem más, mint egy gradiens-módszer segítségével történő numerikus optimalizálás.*

Bizonyítás: *Elegendő azt megmutatnunk, hogy alkalmas célfüggvényt választva a* (3.17) *és* (3.19) *képletekben a* λ *paraméter után szereplő kifejezés nem más, mint az alkalmasan választott célfüggvény súlyonként vett parciális deriváltja. A célfüggvény megválasztásánál a legkisebb négyzetek módszerét alkalmazzuk a tényadatok és neurális háló output adatainak összesimítására:*

$$E = \frac{1}{2}\sum_k (T_k - O_k)^2 \tag{3.20}$$

Nyilvánvaló, hogy E minimalizálásával a legjobban illeszkedő hálózatot alakítjuk ki. Írjuk most fel E-t a súlyok függvényében egy egy rejtett réteggel rendelkező hálózati topológia esetén:

$$E = \frac{1}{2}\sum_k \left(T_k - g\left(\sum_j W_{j,k} a_j \right) \right)^2 = \frac{1}{2}\sum_k \left(T_k - g\left(\sum_j W_{j,k} g\left(\sum_i V_{i,j} I_i \right) \right) \right)^2 \tag{3.21}$$

Képezzük először a $W_{j,k}$ szerinti deriváltakat. Vegyük észre, hogy a_j nem függ $W_{j,k}$-tól, továbbá a j szerinti összegben csupán egyszer fordul elő. Így:

$$\begin{aligned} \frac{\partial E}{\partial W_{j,k}} &= -a_j \left(T_k - g\left(\sum_j W_{j,k} a_j \right) \right) g'\left(\sum_j W_{j,k} a_j \right) = \\ &= -a_j (T_k - O_k) g'\left(\sum_j W_{j,k} a_j \right) \end{aligned} \tag{3.22}$$

Hasonló megfontolások alapján a $V_{i,j}$ szerinti deriváltak a következők:

$$\frac{\partial E}{\partial V_{i,j}} = -I_i g'\left(\sum_i V_{i,j} I_i \right) \sum_k \left(W_{j,k} (T_k - O_k) g'\left(\sum_j W_{j,k} a_j \right) \right) \tag{3.23}$$

Ha összevetjük a (3.22) *és* (3.23) *eredményeit láthatjuk, hogy a* (3.17) *és* (3.19) *képletekben alkalmazott gradiens-módszer gradiens irányait adják. Ezzel az állítást beláttuk.*

■

Sokan kritizálják a gradiens-módszer alapjain működő neurális hálózat tanítását, és ezt meg is érthetjük a 3.2. alpont tanulságai alapján. A tanulás szempontjából kritikus az indulóérték (induló súlyok) és a lépésköz. (Megjegyezzük, hogy az itt λ–val definiált lépésközt a neurális szakirodalom gyakran jelöli η–val és nevezi *bátorsági faktornak* vagy *tanulási tényezőnek – learning rate*. Értékét pedig – a 3.2. pontban elmondottakból egyértelműen következően – nem konstans, hanem a tanulási idő függvényében változó mennyiségnek tekintik.) A tanulási probléma komplexitásával többen is foglalkoztak, és sikerült bebizonyítani, hogy a mintahalmaz optimális közelítésének megkeresése még abban az esetben is NP-teljes probléma, ha igen erős korlátozó feltételeket alkalmazunk,

és ha legalább három processzáló egységet használunk fel. (A pontos tételeket és bizonyításokat ld.: Judd [1990] és Blum, Rivest [1992].)

A súlyok optimalizálásánál is nagyobb problémát jelenthet az optimális hálózati topológia megválasztása. Nyilvánvaló, hogy egy sok neuronból és köztes rétegből álló hálózat súlyoptimalizálása nehezebb, de bonyolultabb összefüggések feltárására is alkalmas. Egy egyszerű hálózatot könnyű tanítani, de reprezentációs képessége is szegényebb. A helyes topológia kiválasztásakor gyakran alkalmazható a genetikus algoritmus is.

A neurális hálózati modellezés esetén néhány szempontot a fentieken kívül érdemes szem előtt tartani. Az egyik legáltalánosabban elfogadott eljárás az, hogy a minta adatbázist három – nem feltétlenül egyenlő – független halmazra osztjuk, mégpedig *tanuló* (*train*), *tesztelő* (*test*) és *validáló* (*validation*) halmazra. Így a súlyok tanulásánál kizárólag a tanuló adatbázis mintáit használjuk, de közben párhuzamosan kiértékeljük a tesztadatbázis eseteit is. Amikor a tanuló adatbázis segítségével már minden további súlymódosítás a tesztelő adatbázis eredményein ront, ott megállítjuk a hálózat tanulását. Elképzelhető ugyanis, hogy egy tanulási mennyiség után már nem egy mintában húzódó általános összefüggéseket fedezi fel a hálózat, hanem a minta specialitását, zajait. Ezt nevezné a hagyományos statisztika *túlillesztésnek*, míg a neurális hálózatokkal foglalkozó szakirodalom *túltanulásként* (*overtraining*) említi. Végül a validáló halmazra szintén meg szokták vizsgálni a hálózat teljesítését, hiszen ezt az adatbázist az optimalizálási folyamat során egyáltalán nem használta fel az algoritmus. Abban az esetben, ha a tanuló és validáló halmazon hasonló eredményeket ér el a hálózat, akkor általában elfogadjuk a végső eredményeket.

3.5. Összefoglalás

Ebben a fejezetben olyan módszereket mutattunk be, amelyek segítségével komplex és sztochasztikus folyamatokat tudunk reprezentálni, illetve optimalizálni. A megoldások hátterében mindig a természet „motiváló ereje”, a tanulás, a fejlődés állt. Ezeket az algoritmusokat hatékonyan alkalmazhatja a közgazdaságtan, hiszen feltehető, hogy a gazdaság szereplői racionálisan viselkednek, és egy időszakban elkövetett nyilvánvaló hibáikat nem ismétlik meg a következő időszakokban. A jó alkalmazhatóság mellett azonban ne felejtsük el, hogy ezek a módszerek sok esetben nem egzaktak. A genetikus algoritmus leállhat lokális optimumban is, a neurális hálóval reprezentált függvény pedig fekete-doboz függvény marad.

4. Fejezet

Az ACE módszertan

Az előző két fejezetben bemutatott módszertan segítségével számos komplex közgazdasági probléma kezelhető. A szimulációs módszertan, a genetikus optimalizálás vagy a neurális hálózat segítségével történő függvényillesztés mind-mind olyan módszertani eszközt biztosít a közgazdászok számára, amelyet külön-külön is be lehet vetni egy adott probléma megoldása során. Arra is utaltunk, hogy e két módszertan szorosan összefügg és számos esetben érdemes ezeket összekapcsolni. Az összekapcsolásnak két lehetőségére szeretnénk felhívni a figyelmet. Az egyik mód, amikor egy szimulációs modellt kívánunk optimalizálni és ehhez evolúciós eljárásokat veszünk igénybe. Ezt a módszert alkalmaztuk például a 2. fejezetben többször is példaként említett opcióárazás modell esetén, ahol az optimális kereskedési stratégiák halmazát kívántuk genetikus algoritmus segítségével megkeresni. A másik mód, amikor a szimuláció egy kisebb eleme önmagában képes evolúciós alapokon működni. Például egy szimulációs modell közgazdasági *szereplőket* (*agent*) modellezhet, és lehetőséget biztosíthat arra, hogy ezek a szereplők saját és társaik hibái alapján tanuljanak, fejlődjenek, azaz az adott gazdasági – vagy társadalmi – környezethez alkalmazkodjanak. Ebben az esetben a modell szereplőit kell ellátni evolúciós módszereket alkalmazó algoritmusokkal. Sőt, egy ilyen modell esetén is fennállhat magának a szimulációnak az egészére vonatkozó optimalizálás, amelyet megint csak megkísérelhetünk evolúciós módszerekkel megoldani.[1]

A sokszereplős szimulációs közgazdaságtan – Agent-based Computational Economics (ACE) – pontosan ezt a két módszertant alkalmazza annak érdekében, hogy komplex közgazdasági rendszerek működését tanulmányozza. Ebben a fejezetben az a célunk, hogy bemutassuk hol tart jelenleg az ACE módszertan, milyen problémákkal foglalkozik, meddig jutott és milyen távlati célokat tűztek ki a módszertan alkalmazói maguk elé. Ezzel megalapozzuk a következő három fejezetet, mivel az ott bemutatásra kerülő modellek mind az ACE módszertanra építenek.

[1] Ennek az igen összetett módszertant alkalmazó lehetőségnek a szemléltetésére vegyük a következő példát. Szeretnénk tervezni egy focipályát, ahonnan probléma esetén a nézők adott számú vészkijáraton távozhatnak. A feladat az, hogyan helyezzük el optimálisan a kijáratokat ahhoz, hogy a közönség a lehető legrövidebb idő alatt távozni tudjon. A szimulációhoz a közönség tagjait olyan szabályalgoritmussal látjuk el, amelyeket neurális hálózatok vezérelnek. A szimuláció többszöri futtatása során kialakulnak a szereplők optimális viselkedési stratégiái, majd ezután egy genetikus algoritmus segítségével meghatározzuk az optimális vészkijárat elhelyezést.

Az ACE módszertan[2] olyan közgazdasági rendszerek számítógép általi vizsgálata, amelyben egymással és/vagy a környezettel kapcsolatban álló, autonóm szereplők tevékenykednek. Ezért az ACE a *komplex adaptív rendszerek* közgazdasági területre történő alkalmazása. (A komplex adaptív rendszerekről ld. részletesen Holland [1992] munkáját). Az ACE módszertan három tudományos terület ötvözéséből született, mégpedig az evolúciós közgazdaságtanból, a számítástudományból és a kognitív tudomány területéből.

Az ACE kutatások egyik köre a decentralizált gazdasági piacokon megfigyelhető globális szabályok, törvényszerűségek kérdéskörét kutatja. Arra keresi a választ, hogy olyan jelenségek, mint például a kereskedelmi hálózatok, a társadalmilag elfogadott pénz, vagy a piaci „játékszabályok" hogyan alakulnak ki – és miképpen változhatnak – minden felülről jövő (*top-down*) szabályozás, illetve tervezés ellenére. A kihívás abban rejlik, hogy konstruktív módon bizonyítani (vagy demonstrálni) tudjuk, hogy hogyan jöhetnek létre ezek a törvényszerűségek alulról (*bottom-up*), az önérdekük által vezérelt autonóm gazdasági szereplők ismételt, egymás közötti (lokális) interakciójával.

Az ACE kutatások másik köre az ACE keretrendszert számítógépes laboratóriumként használja.[3] Ezen kísérletek segítségével számos gazdasági-társadalmi struktúra tanulmányozható és tesztelhető, különösképpen az egyéni viselkedésre és a társadalmi jólétre vonatkozó hatások tekintetében. A vizsgálódás során nem csak a globális szabályszerűségek megfigyelése, hanem mélyebb magyarázatuk a cél. Azaz nem csak az a fontos, hogy miért alakulnak ki bizonyos összefüggések, hanem az is, hogy miért nem alakulnak ki másmilyenek.

Sem a közgazdasági rendszerek *önszerveződő* (*self-organizing*), sem pedig evolúciós rendszerekként történő felfogása nem vadonatúj fogalom a szakirodalomban. A legelső megfogalmazások[4] jóval a számítógép rohamos térhódítása előtt napvilágot láttak, és számos kutatót inspiráltak, új elemzési területeteket hoztak létre (pl.: evolúciós játékelmélet). Miért újszerű akkor az ACE modellezés? Elsősorban azért, mert kiaknázza a számítástechnika által biztosított eszközöket, ezek közül is kiemelve az óriási adattároló és feldolgozó kapacitást, a nagypontosságú számábrázolást, a rendkívül nagy sebességet elérő *párhuzamosítási technikát* és nem utolsó sorban az *objektum-orientált programozást*. Ezek az eszközök négy irányban terjesztik ki az ACE kutatók lehetőségeit.

---

[2] Tesfatsion [2001] munkájára és az ACE módszertant bemutató hivatalos honlapra (http://www.econ.iastate.edu/tesfatsi/ace.htm) építünk.

[3] Számos publikációban a *Petri-csésze* (*Petri-dishes*) kísérletezéshez hasonlítják.

[4] Az önszerveződő közgazdasági rendszer gondolata már Smith [1937] és Hayek [1948] munkáiban megjelenik, míg az evolúciós rendszeré Schumpeter [1942] és Alchian [1950] műveiben.

Heterogén szereplők. A számítógép segítségével épített közgazdasági világokat olyan heterogén szereplők népesítik be, akik a környezettel és más szereplőkkel kapcsolatos interakciójukat, döntéseiket beépített (beprogramozott) társadalmi normák és viselkedési szabályok, valamint a múltban tapasztalt jelenségek és adatok birtokában kialakított viselkedési szabályok alapján határozzák meg. Éppen ezért ezek a szereplők önállóságra és magasabb szintű „gondolkodási" struktúra kiépítésére képesek, mint a hagyományos modellek gazdasági szereplői. Hérdle és Kirman [1994] empirikus vizsgálat segítségével mutatja be a reprezentatív szereplők hiányosságát, és a heterogén szereplők szerepeltetésének fontosságát.

Önszerveződés. Egy ACE modell szimulációja alatt párhuzamosan dolgozhatunk a különböző viselkedésű szereplők széles skálájával, és rendkívül nagyszámú interakciót hajthatunk végre, és figyelhetünk meg ezekben a közgazdasági világokban. Versenyző és kooperáló kapcsolatok, árigazodási mechanizmusok, és számos egyéb jelenség alakulhat ki automatikusan azáltal, hogy a modell szereplői folytonosan továbbfejlesztik viselkedésüket, a rendkívül nagy számú szereplő-környezet és szereplő-szereplő interakciók eredményei hatására, annak érdekében, hogy saját céljaikat megvalósítsák. Úgy is fogalmazhatunk, hogy az önérdek által vezérelt mikro szintű viselkedés egy magasabb szintű társadalmi-gazdasági jelenséghez vezet, amelyet önszerveződésnek nevezünk.

Evolúció. A gazdasági szereplők evolúciós képessége folytán a szereplők viselkedési paramétereire vonatkozóan inkább a természetes szelekció érvényesül, és nem a populáció szintű mozgástörvények. A szelekciós eljárás nyomán a szereplők képesek a múlt tapasztalatai alapján módosítani régi viselkedésükön, illetve új viselkedési szabályokat létrehozni. A szimulált gazdaság és annak szereplői együttesen, egymásra hatva válnak egyre életképesebbé.

Beavatkozás mentesség. Az ACE módszertan segítségével lehetőség van arra, hogy hosszú futási időt biztosítsunk egy-egy szimulált világnak, amelyet megfigyelünk, de ugyanakkor nem avatkozunk be. Amint az induló értékeket és feltételeket a modellező beállította, szabadon engedheti az eseményeket, amelyet ettől kezdve a szereplők egymás közötti és a környezettel kapcsolatos interakciói irányítanak, minden további külső beavatkozás nélkül.

Összefoglalva, az ACE olyan módszertan, amely az evolúciós közgazdaságtan, a kognitív tudományok és a számítástudomány koncepciói és eszközei segítségével

- konstruktív alapokat nyújt – gondolkodó és cselekvő autonóm gazdasági szereplők által – gazdasági elméletekhez,
- megfontolt számítógépes kísérletek eredményeinek statisztikai vizsgálata, továbbá analitikus, ökonometriai, gyakorlati és humán-laboratóriumi eredményekkel történő összehasonlítása által teszteli, finomítja és kibővíti ezen elméleteket.