WinBUGS

1. ábra. Thomas Bayes (1702 – 1761)

*„Given the number of times in which an unknown event has happened and failed: **Required** the chance that the probability of its happening in a single trial lies somewhere between any two degrees of probability that can be named." (Bayes, 1763)*

A bayesi statisztikai elemzésekben általánosan használt szoftver a WinBUGS.[1] Az az ún. BUGS (***B**ayesian inference **U**sing **G**ibbs **S**ampling*) projekt során fejlesztették ezt az eszközt. A projekt célja az volt, hogy komplex statisztikai feladatok bayesi elemzéséhez rugalmas szoftvereket, illetve egy általános célú programozási nyelvet hozzon létre (BUGS-nyelv). A projekt 1989-ben indult az Medical Research Council (MRC) Biostatisztikai Egységében (Cambridge), és kezdetben a 'Classic' BUGS program fejlesztését jelentette. Később a WinBUGS fejlesztésbe bekapcsolódott az Imperial College School of Medicine at St Mary's (London) is. A WinBUGS jelenleg a BUGS-nyelv felhasználásával végzett elemzések stabil eszköze. Emellett egyéb, a BUGS-nyelvet többé-kevésbé kompatibilis módon alkalmazó fejlesztések is léteznek. Egyik ilyen az OpenBUGS fejlesztése,[2] ami a Helsinki Egyetemen (Finnország) folyik. Míg a WinBUGS stabilabb, befejezettnek tekinthető szoftver, az OpenBUGS újabb és újabb funkciókkal egészül ki ennek megfelelően működése esetenként kevésbé megbízható. Míg az OpenBUGS a WinBUGS alapjaira épül, attól eltérő Gibbs-mintavételezést alkalmazó eszközöket is fejlesztenek. Ilyen a C++ nyelven implementált `JAGS`[3]. Ugyancsak C++ nyelven fejlesztett eszköz a `Stan`[4], ami az említettektől eltér a mintavételezési eljárás tekintetében.

Az alábbiakban a szoftver telepítésének, aktiválásának leírása után egy egyszerű példsoron keresztül mutatom be, hogy a WinBUGS segítségével hogyan végezhetünk el bayesi elemzéseket.

[1] `http://www.mrc-bsu.cam.ac.uk/bugs/`

[2] `http://www.openbugs.net/w/FrontPage`

[3] `http://mcmc-jags.sourceforge.net/`

[4] `http://mc-stan.org/`

A program telepítése

A WinBUGS telepítője szabadon letölthető a `http://www.mrc-bsu.cam.ac.uk/software/bugs/the-bugs-project-winbugs/` oldalról.

Ha 32 bites MS Windows környezetben telepítenénk, akkor a `http://www.mrc-bsu.cam.ac.uk/wp-content/uploads/WinBUGS14.exe` fájlt kell letöltenünk és azt a szokásos telepítési varázsló segítségével telepíteni (megadva a telepítési útvonalat).[5]

Ha 64 bites MS Windows rendszerben szeretnénk használni a

[5] Linuxon a Wine-ra telepíthetjük a WinBUGS-ot

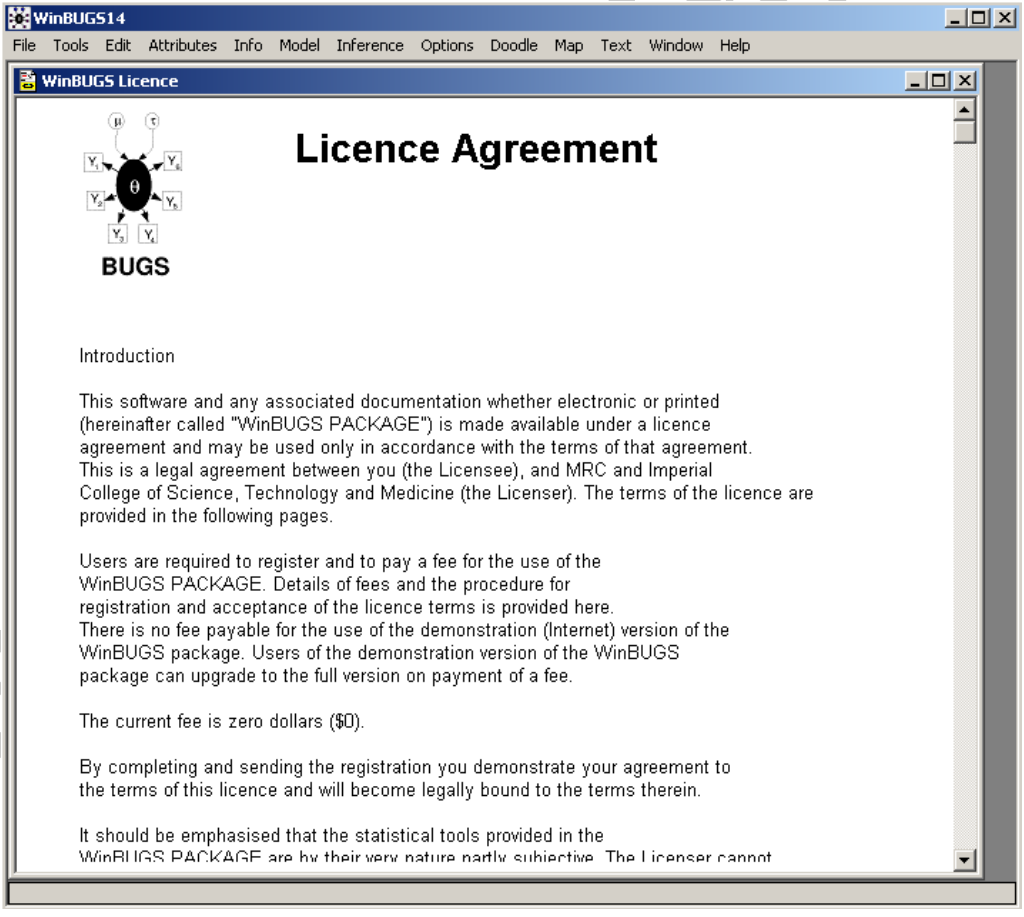
programot, akkor nem tudunk telepítőt letölteni, hanem csak egy ZIP tömörítésű állományt, ami tartalmazza a WinBUGS elemeit[6]. A letöltés után a fájlt tetszőleges helyre tömöríthetjük ki.

[6] http://www.mrc-bsu.cam.ac.uk/wp-content/uploads/winbugs14.zip

Akármelyik telepítési eljárást is alkalmazzuk, sem az Asztalon, sem a programok listájában nem lesz indító ikonunk a program elindításához. Ha nem akarjuk minden alkalommal megkeresni a telepített WinBUGS14.exe-t, akkor érdemes a telepítés után egy parancsikont létrehozni az asztalra.

Fontos tudni, hogy az így telepített program a WinBUGS14, azonban a honlapon elérhető a legfrissebb verzióhoz tartozó „patch", ezen a címen: http://www.mrc-bsu.cam.ac.uk/wp-content/uploads/WinBUGS14_cumulative_patch_No3_06_08_07_RELEASE.txt. Amint a kiterjesztéséből is látható ez egy szöveges állomány. Ennek a WinBUGS-ba való beillesztése nem teljesen magától értetődő, ezért a lépéseit érdemes itt bemutatni.

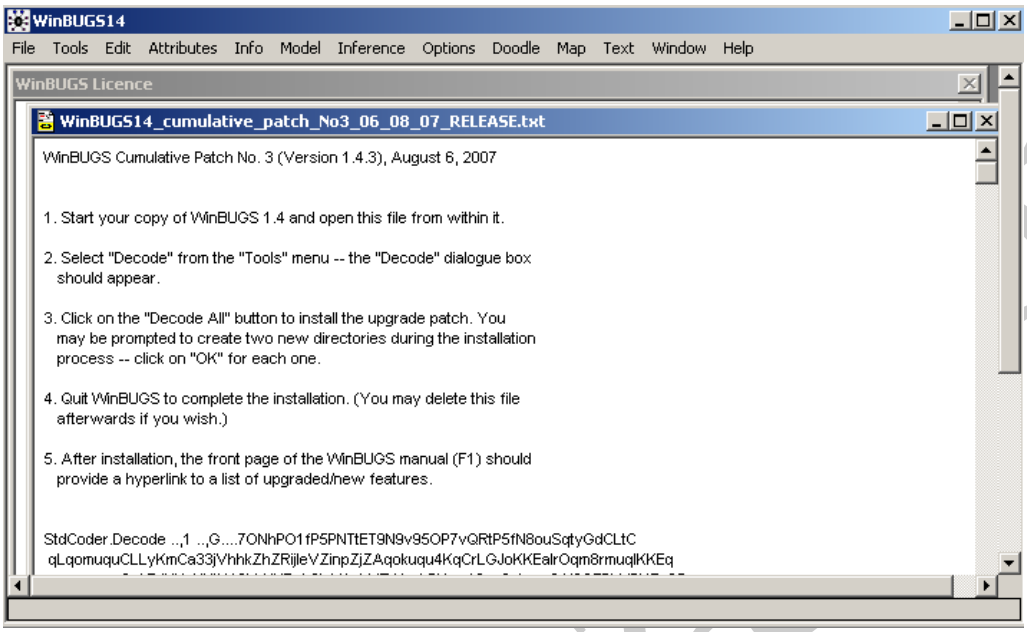
Először töltsük le a patch-fájlt. Majd indítsuk el a WinBUGS-ot, aminek a nyitóképernyőjét mutatja a 2. ábra.



2. ábra. WinBUGS nyitó képernyő

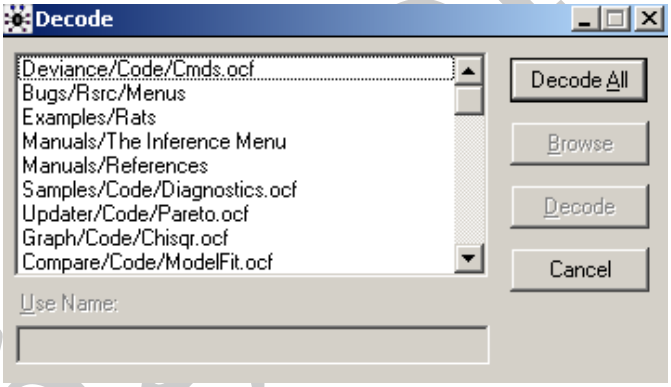
A `File` menü `Open` menüpontjának segítségével nyissuk meg a

patch-fájlt, aminek eredményeként a 3. ábrához hasonlót kell látnunk a képernyőn.



3. ábra. Patch-fájl a WinBUGS-ban megnyitva

A következő lépésben a `Tools` menü `Decode` menüpontjára kell kattintanunk, ennek következtében a 4. ábrán látható ablak jelenik meg.



4. ábra. `Decode`-ablak

Ha a 4. ábrán látható ablakban a `Decode All` gombra kattintunk,[7] akkor a kiegészítések telepítése elkezdődik. Ennek során a program kétszer rákérdez, hogy létrehozzon-e könyvtárakat, mindkét esetben az *igent* válasszuk. A telepítés végeztével a Help menü About WinBUGS menüpontjára kattintva láthatjuk, hogy már az 1.4.3-as legfrissebb verzió fut a gépünkön.

Fontos megemlíteni, hogy ezáltal ugyan a legújabb verzióval rendelkezünk, de még nem tudjuk használni annak minden funkcióját

[7] dekódolás

teljes mértékben. Ahhoz, hogy a programot teljes egészében használhassuk, az előzőekhez hasonlóan decode-olnunk kell a program honlapjáról letölthető kulcsot. Korábban ezt a kulcsot évente újra kellett igényelni, azonban ma már „halhatatlan"[8] kulcsot kapunk a fejlesztőktől (`http://www.mrc-bsu.cam.ac.uk/wp-content/uploads/WinBUGS14_immortality_key.txt`). A decode-olás sikerességét úgy tudjuk ellenőrizni, hogy a WinBUGS telepítési könyvtárában a `Bugs` alkönyvtár `Code` alkönyvtárában megnézzük a `Keys.ocf` létrehozásának dátumát. Ha az a mai nappal egyezik meg, akkor már „halhatatlan" WinBUGS-unk van.

[8] halhatatlanítás

Malacok ivararánya

A malacok vagy hím- vagy nőivarúak[9] lehetnek, így az ivararány statisztikai elemzése során használhatjuk az ún. *binomiális* eloszlást (2. táblázat). Ezt az eloszlástípust használjuk olyan változók vizsgálata során, amelyek az ivarhoz hasonlóan csupán kétféle értéket vehetnek fel (pl. pénzfeldobás, diagnosztikai tesztek).

[9] kan- vagy emsemalac

A binomiális eloszlás két paraméterrel rendelkezik, az egyik paraméter (p) azt a valószínűséget jelenti, amilyen valószínűséggel a változónk az egyik értéket (igen vagy nem, 1 vagy 0, pozitív vagy negatív) veszi fel, a másik paramétere (n) pedig a minta méretét jelenti.

A mi esetünkben mondjuk arra lennénk kíváncsiak, hogy tíz[10] malacból álló almokban az emsék darabszáma milyen eloszlást (5. ábra) mutat akkor, ha azt tételezzük fel (naivan), hogy a két ivar valószínűsége ugyanolyan, vagyis 50-50%. Ezt a binomiális eloszlás p-paraméterére lefordítva, annak a valószínűsége, hogy a megszületett malac emse $p = 0.5$. A bayesi, WinBUGS-os szóhasználatban itt tulajdonképpen *predikciót* fogalmazunk meg az emsék eloszlására vonatkozóan.

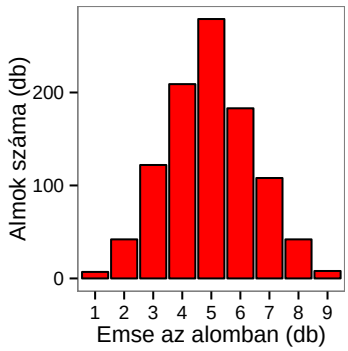
[10] ez lenne az n

A eloszlás WinBUGS-on belüli vizsgálatához egy szkriptet kell írnunk a BUGS-nyelv szabályait követve:

```
model{
  emse.db ~ dbin(0.5, 10)
}
```

Itt meg kell állnunk a kódunk egyes elemeit értelmezni. A BUGS-ban a modellünket kapcsos zárójelek közé kell foglalnunk, aminek nyitó eleme elé a `model` szót szokás írni (mást is lehet).

A modellünk egyetlen sorában egy (össze)függőségi viszonyt fogalmaztunk meg a BUGS-nyelv szerint. A BUGS-modellezésben a modell szerkezeti elemeit node-oknak[11] nevezik. A node-ok közti kapcsolatot pedig gyakran a „szülő-gyermek" (parent-child) kapcsolatként említik. A modellünkben két node van: a `emse.db` és a



5. ábra. Emsék darabszámának prediktált eloszlása tízes almokban

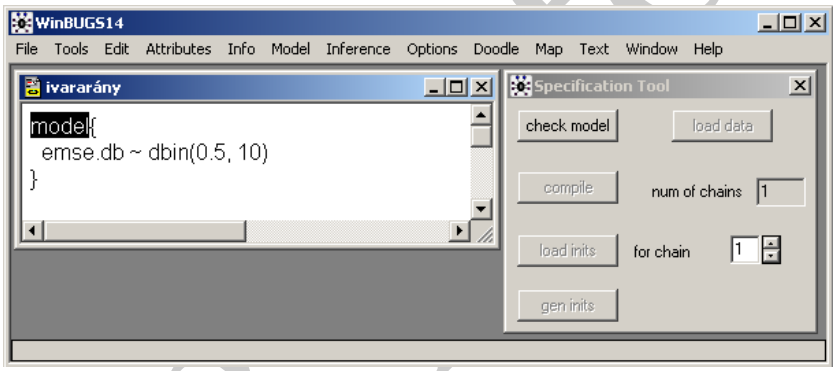
[11] A gráfokkal rokon ez a megközelítés, így nevezhetnénk csúcsoknak vagy csomópontoknak is a node-okat, de szerintem ezek az elnevezések nem fejezik ki a *node* itteni jelentését, így a továbbiakban is ezt az angol elnevezést használom. Ezzel szemben az összefüggéseket nevezhetjük irányított élnek, ugyanúgy, mint a gráfok leírásánál.

`dbin(0.5, 10)`. Ezek között a ~ jel meghatározza az összefüggés természetét. A „szülő-gyermek" node-közötti kapcsolat leírására a BUGS-nyelvben az említett ~ jel mellett még egy másik jel használatos: `<-`. A két jel különböző kapcsolatot jelent, a ~ sztochasztikus kapcsolatot, a `<-` logikai kapcsolatot. A logikai kapcsolat esetén a bal oldalon egy logikai node van, a jobb oldalon pedig valamilyen kifejezés, ami a 3. táblázatbban felsorolt logikai függvényekből állhat. A sztochasztikus kapcsolat során a bal oldalon lévő node egy sztochasztikus node, míg a jobb oldalon lévő node valamilyen eloszlás (2. táblázat).

Miután a modellünk szintaxisát megértettük, itt az ideje, hogy a WinBUGS segítségével elemezzük a modellünk futtatásából származó ivararány-eloszlást. Ehhez nyissuk meg a WinBUGS-ot, és a `File` menüből a `New` menüelem segítségével hozzunk létre egy új dokumentumot és mentsük is el egyből `.odc` kiterjesztéssel.[12] A dokumentumba gépeljük be a modellünket ugyanúgy, ahogy az a kéken szedett szakaszban látjuk.

A `Model` menüből a `Specification...` menüelemre kattintva nyissuk meg a `Specification Tool` ablakot.

Ezek után a dokumentumunkban dupla kattintással jelöljük ki a `model` szót és a Specification Tool ablakban kattintsunk a check model gombra (6. ábra).



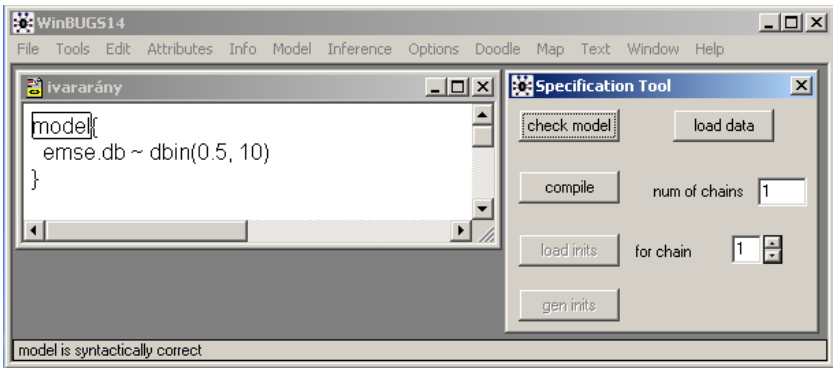
6. ábra. Modell futtatásának előkészítése. A modell szó kijelölése után a `Specification Tool` ablakban a `check model` gombra kell kattintani.

Ha modellünk szintaktikailag rendben van, akkor a WinBUGS ablak alján megjelenik a `model is sintactically correct` üzenet.[13] Emellett a `Specification Tool` ablakban elérhetővé válik a `load data` és a `compile` gomb (7. ábra). Ha lennének betöltendő adataink, akkor azok betöltése lenne a következő lépés (`load data`), de mivel itt most nincsenek ilyenek, kattintsunk a `compile` gombra. Ennek következtében a WinBUGS ablak alján a `model compiled` üzenet jelenik meg, és elérhetővé válik a `load inits` és a `gen inits` gomb (8. ábra). A gen inits gombra kattintva a WinBUGS ablak alján megjelenik az `initial values generated, model initialized` üzenet, ami azt

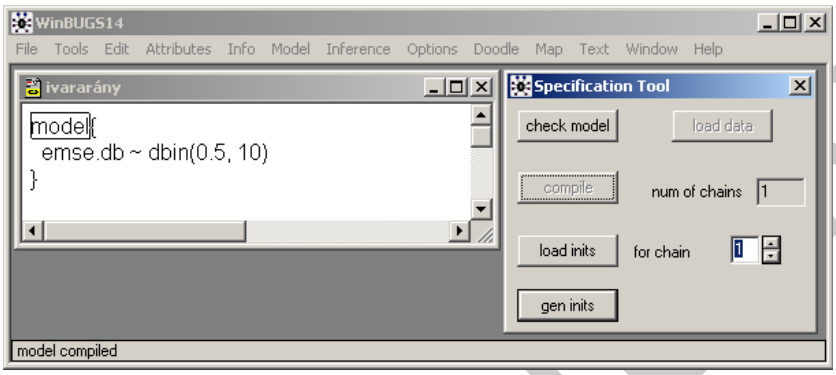
[12] Ez a típusú fájl (compound document) különböző elemeket tartalmazhat majd a későbbiekben, így egyszerű szöveges elemeket, táblázatokat, ábrákat. Ennél fogva a modellezés során használt szkript elemek és forrás adatok mellett az elemzések eredményeit, illetve az azokhoz írt értelmezésinket egy fájlban tárolhatjuk.

[13] A modellben valamilyen szintaktikai hibát ejtettünk, akkor arra utaló üzenetet kapunk az ablak alján.

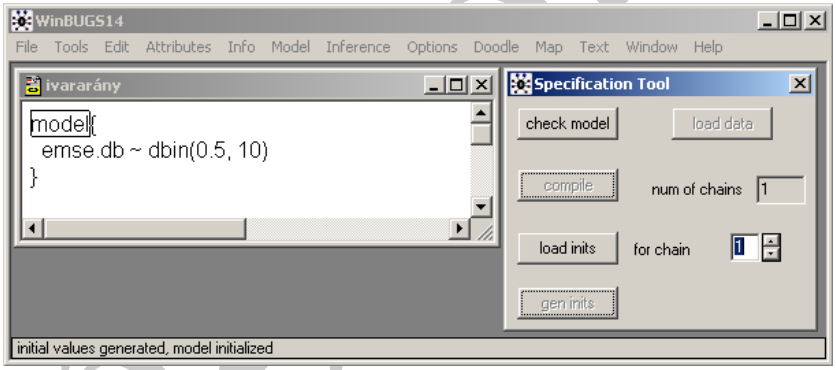
jelzi, hogy a modellünk kész a futtatásra (9. ábra).



7. ábra. A modell futtatásának előkészítése.



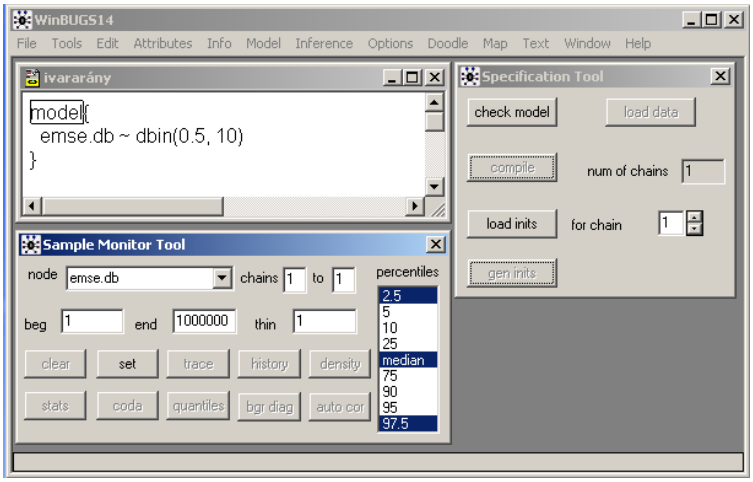
8. ábra. A modell futtatásának előkészítése.



9. ábra. A modell futtatásának előkészítése.

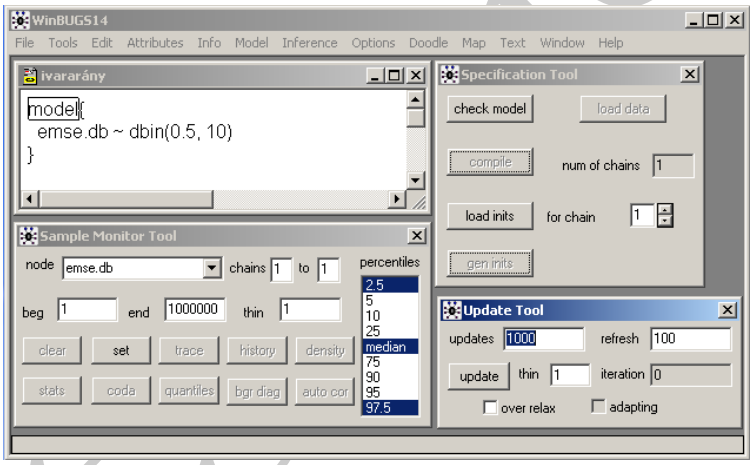
A következő lépés, hogy meghatározzuk azt, hogy a modellünk mely node-jaira vonatkozóan szeretnénk ismereteket szerezni. Ezt úgy tehetjük meg, hogy az `Inference` menüből a `Samples` menüelemre kattintunk, aminek következtében megjelenik a `Sample Monitor Tool` ablak (10. ábra). A vizsgálandó node-okat egyenként a `node` mezőbe kell beírni, majd mindegyik beírása után rá kell kattintani a `set` gombra. Ha hibás node-ot írunk be, akkor a `set` gomb elérhetetlen lesz. Esetünkben írjuk be a „`emse.db`” szót, és kattintsunk a `set`

gombra. További node-ok megadása nem lehetséges.



10. ábra. A modell futtatásának előkészítése.

A modell futtása következik, amihez a `Model` menüből az `Update...` menüelemre kattintva be kell hívnunk az `Update Tool` ablakot (11. ábra). Az alapértelmezett beállításokat használjuk az első futtatásnál, vagyis az `updates` mezőben az 1000 szerepeljen, és kattintsunk az `update` gombra. A futtatás elindítása után az `Update Tool` ablak `iteration` mezőjében 100-as lépésekben láthatjuk, hogy a futás hol tart, amikor eléri az 1000-et, akkor vége.



11. ábra. A modell futtatása.

Ennél a pontnál érdemes megállnunk és átgondolnunk, hogy mit is csinált eddig a WinBUGS. Ahhoz, hogy erről képet kapjunk, valamennyivel messzebbről kell indulnunk.

Először is idézzük fel, hogy mire keresünk választ: szeretnénk tudni, hogy tízes almokban az emsemalacok száma milyen eloszlást mutat (milyen gyakori, hogy nincsen? milyen gyakori, hogy egy van? milyen gyakori, hogy kettő van?, stb.). Ezt az eloszlást, illetve

ennek az eloszlásnak bizonyos tulajdonságait különböző eszközökkel vizsgálhatjuk:

1. A legegyszerűbb módszer a *fizikai kísérlet*, ami ebben az esetben az lenne, hogy nagyon sok 10 malacból álló alomban megszámlálnák az emsemalacok számát. Megszámolnánk, hogy hány alomban nem volt emsemalac, hányban volt egy, hányban kettő és így tovább.

2. Amennyiben a vizsgált változó leírható valamilyen egzakt *algebrai* kifejezéssel, akkor azzal kiszámolhatók az eloszlás tulajdonságai. (Más esetben közelítő eloszlások alkalmazásával számíthatjuk ki.)

3. További lehetőség, hogy *számítógépes szimulációval* szerzünk információt az adott eloszlásról. Nagy vonalakban itt arról van szó, hogy véletlen számok megfelelő függvényeivel nagy mennyiségű mintát veszünk a vizsgált eloszlásból, majd ezek alapján tapasztalati becsléseket tudunk megfogalmazni az eloszlás tulajdonságaira vonatkozóan. Ezt a megközelítést széles körben *Monte Carlo* (MC) -eljárásnak nevezik, és a WinBUGS-ban ezt használva is végezhetünk elemzéseket.

Mit jelent az MC a példánkban? Tulajdonképpen a WinBUGS azt csinálja, hogy egy $p = 0.5$ és $n = 10$ paraméterekkel meghatározható binomiális eloszlásból sokszor, véletlenül vesz ki értékeket. Ezek az értékek az n-nek megfelelően 0 és 10 közötti egész számok lehetnek. Ezek az értékek egy láncot alkotnak. Azt hogy hány ilyen láncból kívánunk dolgozni, a `Specification Tool` ablakban állíthatjuk be (`num of chain` mező), a `compile` gomb megnyomása előtt. A láncok létrejöttéhez a programnak szüksége van egy kezdő értékre (`inits`), amit mi is megadhatunk (`load inits`), vagy egyszerűbb modellek esetén a `gen inits` gomb segítségével generáltathatunk a WinBUGS-al.

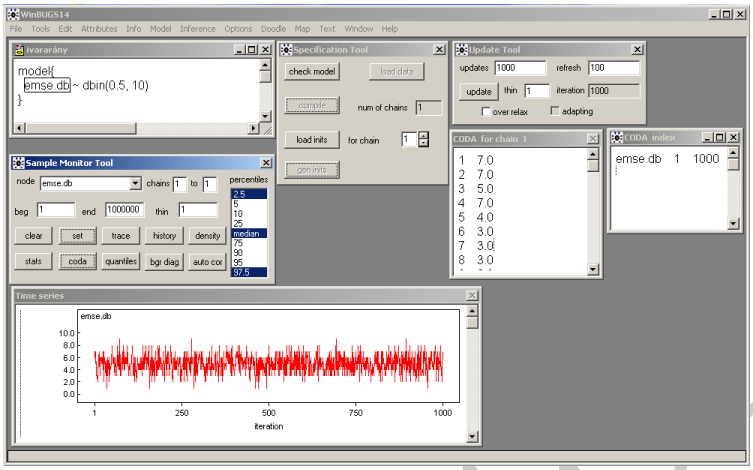
Az `Update Tool` ablak segítségével állíthatjuk be, hogy hány mintát (`updates` mező) vegyen a program az eloszlásból, illetve, hogy a kivett értékekből melyek legyenek a lánc tagjai (`thin` mező). Ez utóbbit úgy kell érteni, hogy ha az értéke 1, akkor minden kivett érték a lánc része lesz, ha 2, akkor csak minden második kivett érték szerepel majd a láncban, ha 3 akkor csak minden harmadik, és így tovább.

Az `Update Tool` segítségével generálódott láncokat a `Sample Monitor Tool` segítségével elemezhetjük. Ehhez először a korábban már használt `node` mezőben meg kell adnunk, hogy melyik node-ra vonatkozóan vizsgálódnánk. Ha egy node-al akarunk csak foglalkozni, akkor annak a nevét kiválasztjuk a legördülő mezőből, ha minddel, akkor pedig $*$[14] jelet írunk a mezőbe. [14] *

A `clear` gomb megnyomásával töröljük a megadott node-ra vonatkozó láncokat.

A `chains` mezőkben megadhatjuk, hogy mely láncokat szeretnénk a vizsgálatokban figyelembe venni, a `beg` és `end` mezőkben azt, hogy a láncok hányadik elemétől kezdve és hányadik eleméig, a `thin` mezőben pedig azt, hogy a láncok hányadik elemeit.

A láncok egyes értékeit mintavételi sorban számszerűen CODA-formátumban a `coda`, grafikusan a `history` gomb megnyomásával kapjuk meg.



12. ábra. Láncok értékei

A CODA két dokumentumban jelenik meg, az egyik egy index-állomány, a másik pedig a lánc elemeit (azok sorszámával) tartalmazza. Ez a formátum lehetőséget ad arra, hogy a láncok nyers értékeit más szoftverekkel (`S-Plus`, `R`) elemezhessük.

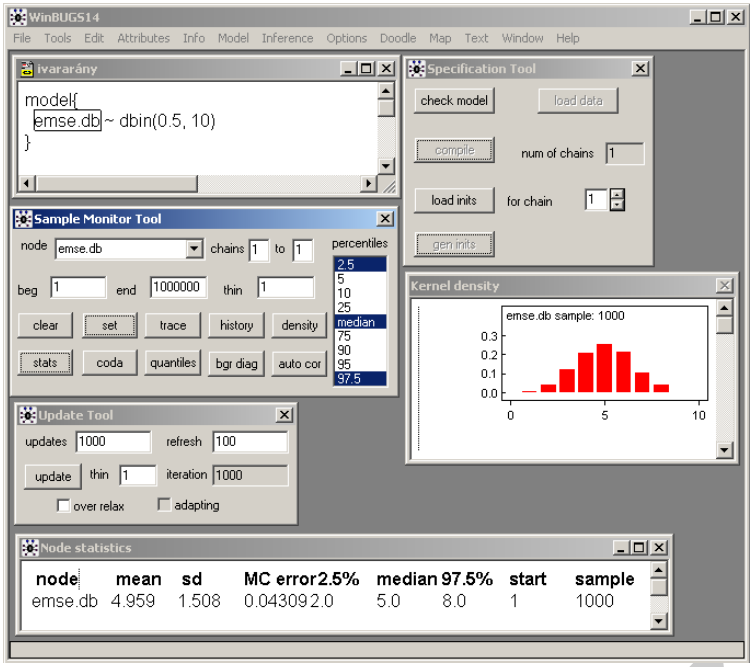
A láncok nyers értékeinek eloszlását a `density` gom megnyomásával kapjuk meg (`Kernel density`). A `stats` gomb pedig ennek az eloszlásnak fontosabb mutatót írja ki a `Node statistics` ablakba.

A 13. ábrán látható `Node statistics` ablak tartalma jobban olvasható formában:

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
emse.db	4.959	1.508	0.04309	2.0	5.0	8.0	1	1000

Az első oszlopban szerepel a node neve, az utolsó két oszlopban az elemzés alapját képező értékek láncbeli kezdő sorszámát és a minták számát láthatjuk. A második és harmadik oszlopban a node megadott tartományában szereplő értékeinek átlaga és szórása olvasható. A negyedik oszlop az átlag MC standard hibája.[15] A következő három oszlop az eloszlás percentiliseit mutatja. Ezeket a `Sample Monotor Tool` `percentiles` listájában adhatjuk meg. Alapértelmezésben a 2.5%-os, az 50%-os és a 97.5%-os percentiliseket írja ki a program.

[15] `MC error`$= \sigma / N^{1/2}$, ahol σ a szórás, N pedig a lánc elemeinek száma



13. ábra. A modell futtatásának eredményei

Ebből a táblázatból olvashatjuk ki azokat az eredményeket, amelyek választ adnak az ivararánnyal kapcsolatos kérdésünkre. Így láthatjuk, hogy az emsemalacok száma egy tízes alomban átlagosan 4.959, aminek a szórása 1.508. Azt is kiolvashatjuk a táblázatból, hogy a kocák száma 95%-os valószínűséggel 2 és 8 közé esik. Nagyon fontos tudnunk, hogy a bayesi statisztikában ezt a tartományt nem *konfidencia intervallum*nak, hanem *credible interval*-nak (CR)[16] nevezzük. A CR azt a tartományt jelenti, amibe (bizonyos valószínűséggel, itt 95%-os) a vizsgált értékünk beleesik (a konfidencia intervallum nem ezt jelenti!!!). A táblázatból a `Kernel density` ábrával együtt vizsgálva az is látható, hogy a medián értéke a legvalószínűbb, vagyis leggyakrabban öt emse születik egy tízes alomban.

Ugyanakkor azt is kiolvashatjuk az eredményekből, hogy 2.5% a valószínűsége annak, hogy kettőnél kevesebb vagy nyolcnál több emse legyen egy tízes alomban.

Most, hogy a példánkban feltett első kérdésünket megválaszoltuk, mielőtt további kérdésekre keresnénk (összetettebb modellek segítségével) választ, a WinBUGS egy sajátos moduljáról, a `Doodle`-ről[17] ejtsünk néhány szót. A `Doodle` segítségével grafikusan hozhatunk létre node-okat, illetve azok között függőségi viszonyokat. Próbáljuk ki, hogy hogyan tudjuk létrehozni az eddig vizsgált modellünket ezzel az eszközzel. A WinBUGS `Doodle` menüjéből a `New...` menüelem segítségével hozzunk létre egy új dokumentumot. A `New...` elemre kattintva a dokumentum megjelenése előtt egy beállítási ablak

[16] Azért, hogy rövidítésében is elkülönítsük a konfidencia intervallumtól, aminek a rövidítése CI, a credible interval-t CR-ként fogjuk jelölni

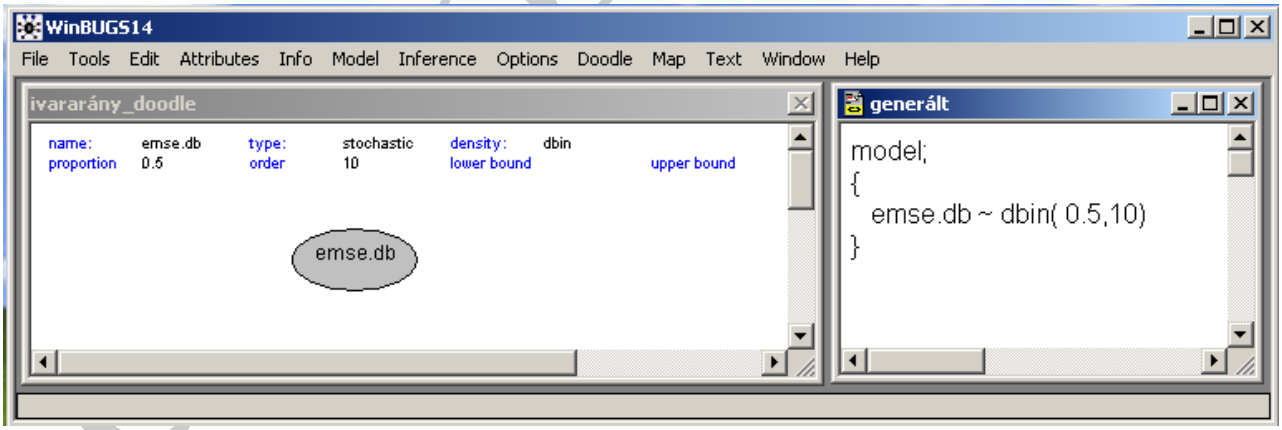
[17] Doodle

(`New Doodle`) jelenik meg, amiben megadhatjuk, hogy milyen széles és magas legyen a szerkesztő ablakunk, illetve, hogy a node-ok milyen szélesek legyenek. Ha az `ok` gombbal elfogadtuk a beállításokat és megjelent az új dokumentumot mentsük is el `.odc` kiterjesztéssel.

Az első node létrehozásához kattintsunk az egérrel bárhova az új dokumentum fehér felületére.[18] Ekkor megjelenik egy szürke ellipszis és az ablak felső részén több mező, amivel meghatározhatjuk a node tulajdonságait (14. ábra).

A `name` mezőbe be kell írnunk a node nevét. Hogy példánkkal összhangban legyen, nevezzük `emse.db`-nek. A következő lépés, hogy a `type` szóra kattintva kiválasszuk a node típusát. Ez `stochastic`, `logical` vagy `constant` lehet. A kiválasztott típusnak megfelelően az ablak felső részén lévő feliratok változnak. A példához igazodva a sztochasztikus típust válasszuk ki. Ezután a `density` szóra kattintva a `dbin` eloszlást válasszuk ki. A kiválasztott eloszlástípusnak megfelelően az ablak felső részén lévő, a node meghatározását segítő feliratok (mezők) változnak. A `proportion` szóra kattintva megjelenő kurzorhoz írjuk be a korábbi p-értéket, a 0.5-öt. A `order` szóra kattintva megjelenő kurzorhoz írjuk be a korábbi n-értéket, a 10-et. Ezzel kész is a példának megfelelő grafikus modellünk. A programmal a létrehozott grafikus modellből szkriptet generáltathatunk a `Doodle` menü `Write code` elemére kattintva (14. ábra). A `Doodle` egyszerűbb modellek szerkesztésénél, illetve a modell szerkezetének vizualizálására hasznos eszköz lehet, de összetettebb modellek létrehozására sok esetben nem alkalmas.

[18] Ha véletlenül több node-ot hoztunk létre, mint amennyit használunk, akkor a kijelölt node-ot törölhetjük a `CTRL+Delete` vagy a `CTRL+Backspace` billentyűkombinációval



14. ábra. Doodle-szerkesztő egy node-al és generált szkripttel

Malacok ivararánya II.

Az előző modellünkből számos fontos ismerettel gazdagabbak lettünk,[19] azonban látva az eredményeket újabb kérdések merülhetnek

[19] *Bayesian knowledge update*

fel bennünk. Pl. mi a valószínűsége annak, hogy egy tízes alomban legalább hét emsemalac születik?

Ennek megválaszolására az előző modellünket kiegészítjük egy újabb node-al, mégpedig az eddigi sztochasztikus node-unk mellé egy logikai node-al:

```
model{
  emse.db ~ dbin(0.5, 10)
  valoszinu7 <- step(emse.db - 6.5)
}
```

A modellünk első sora nem változott, a második sorában viszont logikai kapcsolatot jelez a `<-` jel, a jobb oldalán pedig egy logikai függvény, a `step()` szerepel. A `step()` függvényben argumentumként egy olyan számítást adunk meg, aminek az eredményéről azt akarjuk tudni, hogy nullánál nagyobb vagy sem. Ha a zárójelben lévő számítás eredménye kisebb, mint nulla, akkor a `step()` függvény nulla értéket ad vissza, ha nagyobb akkor pedig egyet. Ennek megfelelően ha `emse.db` node értékéből kivonunk 6.5-öt,[20] akkor csak akkor kapunk vissza a `step()` függvényből egyes értéket, ha az nagyobb volt, mint 6.5. Mivel a `emse.db` 0 és 10 közötti értékeket vehet csak fel, így a 7, 8, 9, 10 esetén ad a `step(emse.db - 6.5)` függvény egyet, egyébként nullát illeszt a `valoszinu7` node-ba. Az előzőekben bemutatott lépések[21] után a futtatás eredménye 1000 minta alapján:

[20] lehetne 6.1, 6.2, . . . , 6.9 is

[21] check model, compile, gen inits, node-ok beállítása, update, stats

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
emse.db	4.959	1.508	0.04309	2.0	5.0	8.0	1	1000
valoszinu7	0.152	0.359	0.01158	0.0	0.0	1.0	1	1000

Ha összehasonlítjuk az előző modellünk eredményével ennek a táblázatnak az első sorát, akkor látható, hogy teljes mértékben megegyeznek a becslések. Arra vonatkozóan, hogy mi a valószínűsége annak, hogy egy tízes alomban hét vagy több emse lesz, a modellünk alapján azt mondhatjuk, hogy 15%. Megjegyezve, hogy meglehetősen nagy bizonytalansággal mondhatjuk ezt, mivel a szórás több, mint a kétszerese a becslésünknek. A szimulációval kapott becslést érdemes összevetni az algebrai megközelítéssel kapott eredménnyel:

$$P(7 \text{ vagy több emse}) = \binom{10}{7}\left(\frac{1}{2}\right)^7\left(\frac{1}{2}\right)^3 + \binom{10}{8}\left(\frac{1}{2}\right)^8\left(\frac{1}{2}\right)^2 + \binom{10}{9}\left(\frac{1}{2}\right)^9\left(\frac{1}{2}\right)^1 + \binom{10}{10}\left(\frac{1}{2}\right)^{10}\left(\frac{1}{2}\right)^0$$
$$= \frac{120+45+10+1}{2^{10}} = \frac{176}{1024} = 0.172$$

Látható, hogy ez 17%-os valószínűséget mutat annak a valószínűségére vonatkozólag, hogy 7 vagy annál több emse van egy tízes alomban. Habár szakmailag nincsen jelentősége a két százalékpontos eltérésnek mégis érdemes itt megemlíteni, hogy a szimulációs

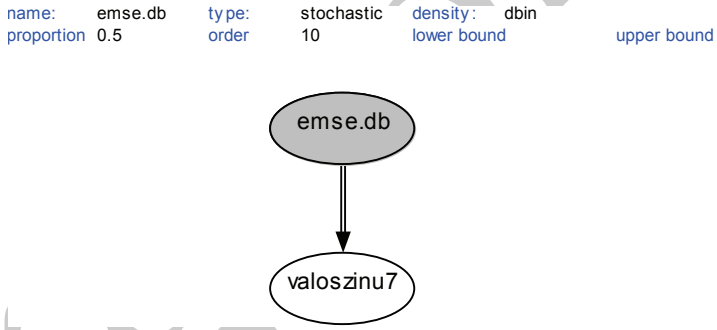
eredmények megbízhatósága annál nagyobb, minél nagyobb számú mintából számítódnak. Ha ugyanazt a modellt nem 1000 mintavétellel futtatjuk, hanem 10000-el, akkor az alábbi eredményeket kapjuk:

```
node          mean      sd   MC error   2.5%   median   97.5%   start   sample
emse.db      5.003    1.56    0.01446    2.0      5.0     8.0       1    10000
valoszinu7  0.1691  0.3748   0.003931    0.0      0.0     1.0       1    10000
```

Összehasonlítva az 1000 mintás futtatással, látható, hogy az MC-hiba jelentősen csökkent, ami érthető, mivel annak számításában a nevezőben benne van a minta mérete (N). A `emse` node-ra vonatkozóan az átlag és a szórás némileg megváltozott, de a 95%-os credible interval nem. A hét vagy annál több emse valószínűségére vonatkozó becslés a 10000 minta alapján már csak három ezrednyi különbséget mutat az algebrai megközelítéssel kapott eredményhez viszonyítva, úgy, hogy a szórás nem csökkent.

Ebben a modellben[22] már két node-unk van, amelyek között függőségi kapcsolat van. A `Doodle` szerkesztőjében az előző node mellett hozzunk létre még egy node-ot, amit nevezzünk `valoszinu7`-nek. A `type` szóra kattintva válasszuk ki a `logical` típust a listából. A `link` beállítását hagyjuk az alapértelmezett `identity` értéken. A `value` szóra kattintva megjelenő kurzorhoz írjuk be a logikai függvényünket: `step(emse - 6.5)`.

[22] Doodle



15. ábra. Doodle-szerkesztő két node-al

A `Doodle` szerkesztőben a node-ok között úgy építhetünk fel kapcsolatot, hogy a „gyermek”-re kattintunk majd a `CTRL` billentyűt lenyomva tartva a „szülő” node-ra kattintunk. A kapcsolat törlése ugyanígy történik. Esetünkben kattintsunk a `valoszinu7` node-ra, nyomjuk le és tartsuk lenyomva a `CTRL` billentyűt és kattintsunk az egérrel a `emse` node-ra. Ennek eredményeként a két node között egy *irányított él* rajzolódik ki (15. ábra). A korábbiakban leírtak szerint a grafikus modellből BUGS-kódot generálhatunk.

Malacok ivararánya III.

A példánkat úgy folytatjuk, hogy adatokat is betöltünk az elemzéshez, ennek megfelelően az alábbiak szerint módosul a szkriptünk:

```
model{
  emse.db ~ dbin(emse.rata, alom.meret)
  valoszinu7 <- step(emse.db - 6.5)
}

# adatok
list(
  emse.rata=0.5, alom.meret=10
)
```

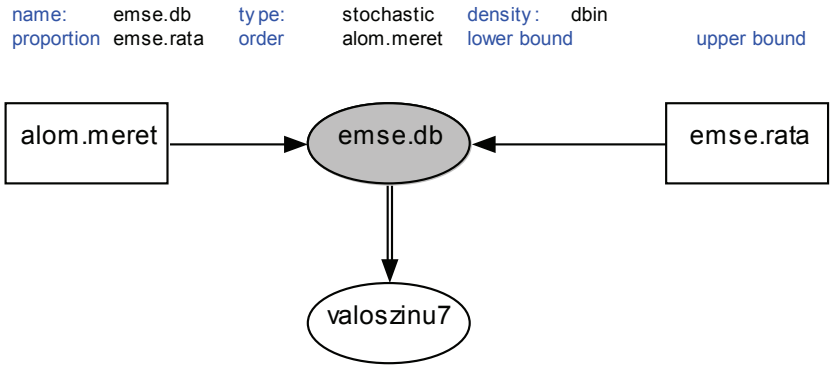
A `emse` node eloszlását leíró `dbin()` függvény paraméterértékeit nem közvetlenül írjuk a zárójelek közé, hanem paraméterenként egy-egy változó nevét írjuk a helyükre. A modell többi része változatlan. Azonban látható, hogy az `adatok` szó alatt egy ún. *lista* kezdődik, amiben adatokat adhatunk meg a program számára. A `list()` függvény használata során fontos, hogy a `list` szó és a `(` között nem lehet szóköz! Vegyük észre azt is, hogy míg a `model` után kapcsos zárójelek közé írjuk a modellünket, addig a `list` után sima zárójelek vannak. A listán belül az egyes változókat vesszővel választjuk el egymástól. A szkriptben látható `# adatok` sor megjegyzés, és mivel a sor `#` jellel kezdődik, a program kihagyja a kód értelmezése során.

A modell futtatása a korábbiakhoz képest annyit változik, hogy a `check model` után az egér segítségével ki kell jelölnünk a `list` szót és a `load data` gombra kell kattintanunk. Ezután ugyanazokat a lépéseket kell végrehajtanunk, mint korábban.[23] A futtatás eredményeként kapott táblázat megegyezik a korábbi eredményekkel.

A `Doodle` szerkesztőben a modellünket úgy változtatjuk meg, hogy két *konstans* node-ot adunk hozzá.[24] Ehhez a szerkesztőben helyezzünk el két új node-ot, majd a `name` mezőben nevezzük el őket és a `type` szóra kattintva válasszuk ki a listából a `constant` típust mindkettőnél. A konstans típus választásánál az addig ellipszis alakú node téglalappá alakul. Mindkét node-ot kapcsoljuk a `emse` node-hoz, „szülői” pozícióban (16. ábra). Fontos, hogy milyen sorrendben kapcsoljuk a konstans node-okat a sztochasztikus node-hoz. Amelyik elsőként kapcsoljuk hozzá, az lesz a sztochasztikus node első paraméterének (esetünkben `proportion`) értéke, amelyik másodikként, az lesz a sztochasztikus node második paraméterének (esetünkben `order`) értéke. Ha elrontottuk a sorrendet, akkor a `proportion` és az `order` szavakra kattintva listából kiválaszthatjuk a megfelelő konstans node-ot.

[23] vagyis: `compile`, `gen inits`, node-ok beállítása, `update`, `stats`

[24] A konstans node-oknak csak nevük és típusuk van, egyéb paraméterrel nem rendelkeznek, illetve mindig *„szülői”* pozícióban lehetnek csak.



16. ábra. Doodle-szerkesztő négy node-al

A 16. ábrán vegyük észre, hogy a konstans nodok és a sztochasztikus node közötti nyíl szára egy vonalból áll, míg a sztochasztikus node és a logikai node közötti nyíl dupla szárú. Ez utóbbi a determinisztikus kapcsolatra utal. A generált szkriptben a megfelelő helyeken vannak az új konstansaink, azonban az azok értékét megadó listát nekünk kell a szkriptbe írnunk.

Malacok ivararánya IV.

Eddig előzetes elképzelésünk[25] alapján úgy gondoltuk, hogy 50% annak a valószínűsége, hogy egy újszülött malac emse lesz. Ez alapján becsültük a tízes almokban az emsék gyakoriságának eloszlását.[26]

[25] *a priori* ismeret

[26] *a posteriori* ismeret

Azonban Tamás barátom, aki sertéstelepen dolgozó állatorvos, azt mondta, hogy ő úgy érzi, hogy általában valamennyivel kevesebb emsemalac születik, mint kan. Mivel épp szabadságon volt, amikor erről beszéltünk számszerűleg ugyan nem tudja megmondani, de gyakorlati tapasztalatai alapján ez a véleménye.

Tehát egy újabb kérdés merül fel, milyen is valójában az emsemalacok születésének valószínűsége? Mivel nem tudott adni a saját állományából származó adatokat, keresgéltem a szakirodalomban. Lamberson et al. (1988) vizsgálatában 1187 újszülött malacból 540 volt emse. Természetesen, jobb lenne ha a cikkben közölt adatokat almonként is ismernénk, de csak összesített adatok érhetők el. Ezek alapján végezhetünk egy bayesi elemzést a nőivar születéskori valószínűségének becslése céljából.

A bayesi elemzések egyik fontos tulajdonsága, hogy előzetes (*a priori*) ismeretek, elképzelések, vélemények, hitek kombinálhatók a megfigyelt adatokkal. Esetünkben a megfigyelt adat az 540 emse az összes 1187 újszülött malacból. Az *a priori* ismeretünk ellentmondásos, mivel eredetileg (az íróasztal mellől) úgy gondoltuk, hogy az emsék ugyanolyan valószínűséggel születhetnek, mint a kanok. Azután viszont azt hallottuk egy gazdálkodó állatorvostól, hogy kevesebb

emse születik, mint kan. A WinBUGS-ban az *a priori* ismeretünket, valamilyen az ismeretet jól leíró prior eloszlással adjuk meg.

Az előző modellünkben a `emse.db ~ dbin(emse.rata, alom.meret)` részt *likelihood*nak is nevezik. A bayesi elemzések során a *likelihood* és az *a priori* ismeretet kombinálva kapjuk meg az *a posteriori*[27] eloszlást. A WinBUGS ebből a poszterior eloszlásból vesz mintákat,[28] e mintákból állnak össze az elemzések alapját képező *láncok*.

[27] inkább a *poszterior* jelzőt fogjuk használni a továbbiakban

[28] Gibbs samplig

A WinBUGS nagy rugalmasságot biztosít a különböző likelihoodok és prior eloszlások kombinálásában. Fontos tudni, hogy vannak olyan likelihood és prior eloszlás kombinációk, amelyekből ún. zárt formában létrehozható a poszterior eloszlás, ezeket a priorokat az adott likelihood konjugált priorjainak nevezik.[29] Ha valamely modellünkben csak konjugált priorokat használunk, akkor a WinBUGS csak egyszerű MC-t használ a poszterior eloszlásból való mintavételhez. Azonban vannak olyan esetek, amikor nem konjugált priorokat kell használnunk, ekkor a WinBUGS *Markov-lánc Monte Carlo (Markov Chain Monte Carlo, MCMC)* eljárást alkalmaz a poszterior eloszlásból való mintavételezésre. Habár a felhasználó ebből gyakorlatilag semmit sem vesz észre, mégis fontos tudnunk erről, mivel az MCMC-eljárást alkalmazó modellek eredményeinek értékelése során (az előzőeken túl) további részletekre is figyelni kell.

[29] likelihoodok és konjugált priorjuk listáját lásd az 1. táblázatban

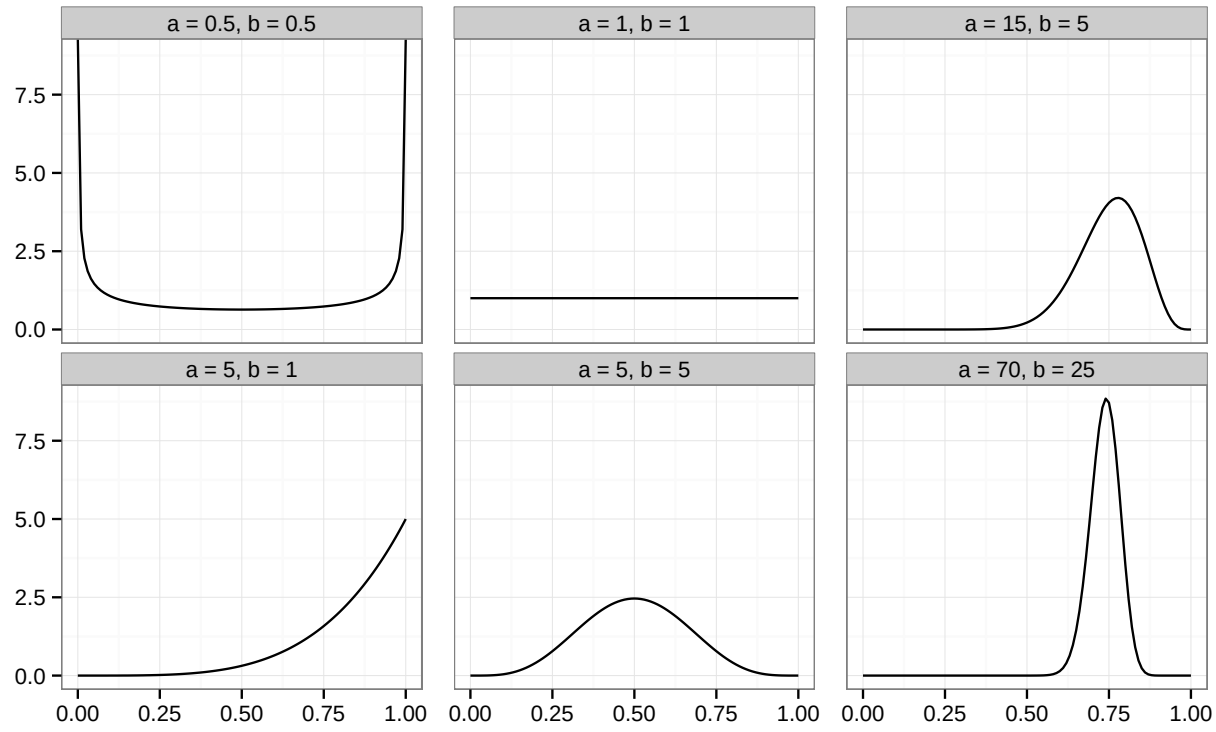
Az ivararány vizsgálatában alkalmazott binomiális likelihood konjugált prior eloszlása a *béta* eloszlás (2. táblázat). A béta eloszlás egy nagyon rugalmas eloszlás, olyan esetben alkalmazzák, amikor valamilyen mennyiség 0 és 1 közé esik. Két paraméterének[30] változtatásával ebben a tartományban nagyon sokféle eloszlást tudunk leírni (17. ábra).

[30] `dbeta(a, b)`

A 17. ábrán középen felül látható $a = 1$ és $b = 1$ paraméterértékek leírható eloszlása a 0 és 1 közötti tartományba eső minden értéknek egyforma valószínűséget mutat. Az ilyen eloszlásokat ún. *nem informatív priorok*ként[31] gyakran alkalmazzák. Leginkább olyan esetben, amikor nem ismert, hogy a lehetséges értékek tartományában mely értékeknek, szakasznak van nagyobb előfordulási valószínűsége. A nem informatív priorok[32] alkalmazása esetén a poszterior eloszlást a likelihoodban használt ún. *kemény információ*k, adatok fogják dominálni. Míg, hogyha valamilyen nagyon határozott prior eloszlást használunk, mint pl. a $a = 70$ és $b = 25$ paraméterű eloszlás (17. ábra), akkor kevés kemény adat mellett a prior jelentős hatással van a poszterior eloszlásra.

[31] flat, non informative prior, vague prior, uniform

[32] valójában minden prior informatív, ezért vannak, akik ehelyett azt javasolják, hogy bizonytalan vagy diffúz priornak nevezzük az ehhez hasonló priorokat



17. ábra. Béta-eloszlások különböző a és b paraméterértékek mellett

Az emsemalacok születéskori arányának becslésére első lépésben használjunk egy olyan modellt, amiben a `emse.rata` priorja egy nem informatív béta-eloszlás.

```
model{
  # likelihood
  emse ~ dbin(emse.rata, alom.meret)

  # prior
  emse.rata ~ dbeta(1, 1)
}

# adat
list(
  emse=540, alom.meret=1187
)
```

A BUGS-nyelv deklaratív nyelv, így mindegy, hogy a modellünkben milyen sorrendben írjuk az egyes node-okat. Van, aki a modell elején írja a priorokat, van, aki pedig a végén, a futtatás eredményét ez nem befolyásolja.

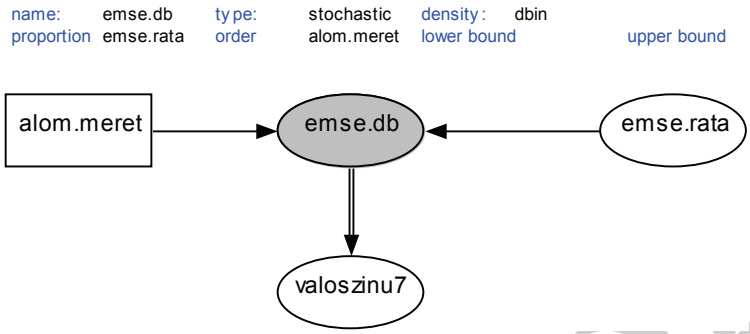
A szokásos lépések[33] után a poszterior eloszlásból vett minták alapján az alábbi eredményeket kapjuk:

[33] check model, load data, compile, gen inits, node-ok beállítása, update, stats

```
node          mean      sd    MC error    2.5%    median   97.5%   start   sample
emse.rata   0.4553  0.0145   1.495E-4   0.4267    0.4552   0.4841       1    10000
```

A táblázat alapján úgy tűnik, hogy 45% körüli az emsemalacok hányada, valamint, hogy ez 95%-os valószínűséggel 43-48% közti tartományban van.

A `Doodle`-szerkesztőben a béta prior eloszlás „szülői” viszonyban lesz a `emse.rata`-hoz viszonyítva. A prior két paraméterének `a`-nak és `b`-nek egyet adjunk meg.



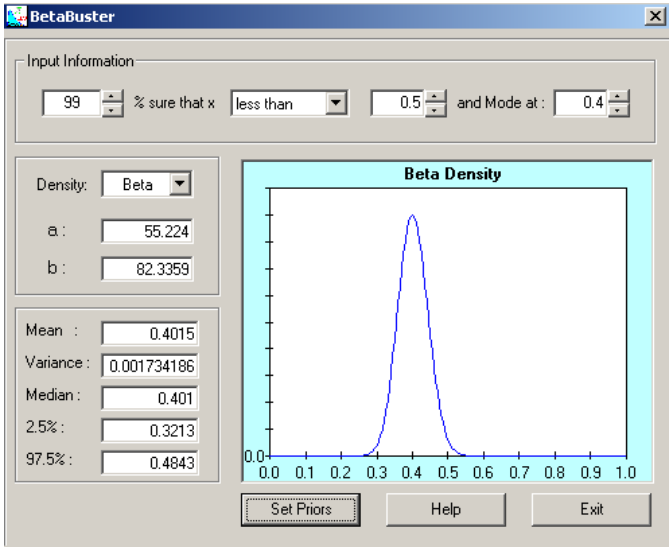
18. ábra. Doodle-szerkesztő három node-al

Most végezzük el a Lamberson et al. (1988) által közölt adatok elemzését úgy, hogy Tamás sejtését megpróbáljuk valamilyen béta-eloszlásban megfogalmazni. Mivel ő úgy gondolja, hogy a malacoknak kevesebb, mint a fele emse, ezért nekünk mondjuk egy olyan eloszlásra lenne szükségünk, aminek a leggyakoribb értéke a 40% és 99%-os valószínűséggel 50% alatt előforduló értékeket tartalmaz csak. Ahogy a 17. ábrán látható, a béta-eloszlás paramétereit változtatva különböző eloszlásokat kaphatunk. Most arra lenne szükségünk, hogy az imént megfogalmazott eloszlás a és b paramétereit meghatározhassuk. Ebben segítségünkre lehet a `Betabuster` szoftver.[34] Ennek segítségével tehát a béta-eloszlások paramétereit határozhatjuk meg. A paramétereket az `R` `epiR`-csomagjának `epi.betabuster()`-függvényével is meghatározhatjuk.[35]
A modellünk az új béta priorral:

[34] `http://www.ics.uci.edu/~wjohnson/BIDA/betabuster.zip`

[35] `epi.betabuster(mode=0.40, conf=0.99, greaterthan=F, x=0.50)`

```
model{
  emse ~ dbin(emse.rata, alom.meret)
  emse.rata ~ dbeta(55.224, 82.3359)
}
list(
  emse=540, alom.meret=1187
)
```



19. ábra. A Betabuster szoftverrel olyan béta-eloszlás paramétereit határoztuk meg, amelynek a módusza 0.4 és 99%-a 0.5 alatt van.

A modell futtatásának[36] eredménye:

[36] check model, load data, compile, gen inits, node-ok beállítása, update, stats

```
node         mean        sd  MC error    2.5%   median   97.5%  start  sample
emse.rata  0.4496  0.01371  1.419E-4  0.4224   0.4496  0.4767      1   10000
```

Az előző „nem informatív” priorral végzett elemzés eredményével összevetve ezeket az eredményeket az alábbiakat fogalmazhatjuk meg. A flat prior esetén az emsék születési valószínűsége 46%, míg a Tamás-féle priorral 45%. A 95%-os credible intervallum gyakorlatilag megegyezik a két különböző priorral végzett elemzésben. Ezek alapján azt mondhatjuk, hogy habár nagyon eltérő priorokat használtunk azok a „kemény adataink” mellett nem dominálták az eredményeket.

Malacok ivararánya V.

A következő elméleti kérdésünk az, hogy az előző példában becsült emsemalac születési ráta alapján mi a valószínűsége, hogy négy vagy annál kevesebb emsemalac születik egy tizennégy malacból álló alomban. A modell az alábbiak szerint bővül:

```
model{
  emse.db ~ dbin(emse.rata, malac.db)
  emse.rata ~ dbeta(55.224, 82.3359)
  emse.pred ~ dbin(emse.rata, alom.meret)
  emse4.valoszinu <- step(4.5 - emse.pred)
}
list(
  emse.db=540, malac.db=1187, alom.meret=14
)
```

Ebben a modellben a becsült `emse.rata` node alapján predikciót végzünk arra vonatkozóan, hogy tizennégyes almokban milyen az emsék darabszámának eloszlása. A modell futtatásának eredménye az alábbi táblázatban olvasható:

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
emse.pred	6.31	1.885	0.02031	3.0	6.0	10.0	1	10000
emse.rata	0.4492	0.01374	1.368E-4	0.4219	0.4493	0.476	1	10000
emse4.valoszinu	0.1687	0.3745	0.003886	0.0	0.0	1.0	1	10000

A táblázat alapján azt gondolhatjuk, hogy egy tizennégyes alomban átlagosan hat emsemalac születik, és a számuk 95%-os valószínűséggel három és tíz közé esik. Annak a valószínűsége, hogy egy tizennégyes alomban négy vagy annál kevesebb malac szülessen: 17%.

Malacok ivararánya VI.

Bocian et al. (2012) közleményében azt mutatta be, hogy a vizsgált almokban (1. napon) 73 emse- és 60 kanmalac született. Ez az arány látszólag eltér Lamberson et al. (1988) eredményeitől, illetve Tamás barátom tapasztalataitól. Ha az előző modellekben a Lamberson et al. (1988) adatait Bocian et al. (2012) adataival helyettesítjük és lefuttatjuk a modelleket, akkor a következő eredményeket kapjuk:
A flat priorral:

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
emse.pred	7.687	1.96	0.01913	4.0	8.0	11.0	1	10000
emse.rata	0.548	0.04285	4.085E-4	0.4628	0.5487	0.631	1	10000
emse4.valoszinu	0.0531	0.2242	0.002154	0.0	0.0	1.0	1	10000

A modellnek ezzel a priorral való futtatása alapján a nőstények születésének a valószínűsége 55%, arányuk 95%-os valószínűséggel 46-63%-os tartományba esik. Egy tizennégyes alomban az emsék száma átlagosan 8 (7.7), illetve 95%-os valószínűséggel 4 és 11 közé esik. Annak a valószínűsége, hogy négy vagy annál kevesebb emse születik, 5%.
Tamás priorjával:

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
emse.pred	6.662	1.918	0.02061	3.0	7.0	10.0	1	10000
emse.rata	0.4735	0.03057	3.091E-4	0.4139	0.4738	0.5335	1	10000
emse4.valoszinu	0.1292	0.3354	0.003347	0.0	0.0	1.0	1	10000

Ezzel a priorral való futtatás alapján az emsék születésének a valószínűsége 47%, arányuk 95%-os valószínűséggel 41-53%-os tartományba esik. Egy tizennégyes alomban az emsék száma átlagosan 7 (6.7),

illetve 95%-os valószínűséggel 3 és 10 közé esik. Annak a valószínűsége, hogy négy vagy annál kevesebb emse születik, 13%.

Hiperpriorok

Ahogy korábban láttuk, a bayesi elemzésekben előzetes ismereteket is bevonhatunk a számításokba. Ha egy adott tudományos kérdéssel kapcsolatban több vizsgálatból is származnak adatok, akkor azt gondolhatnánk, hogy ezek a vizsgálatok a bayesi tanulási folyamatban egy lánc szemei lehetnek. Vagyis az egyik vizsgálatból származó eredmény (poszterior eloszlás) egy következő vizsgálat priorjává válhat. Eddig csak olyan elemzéseket végeztünk, amikor a prior vagy flat vagy hipotetikus, mért adatok nélküli volt.

Az előző példa után felmerülhet bennünk az a kérdés, hogy hogyan lehet egy elemzés poszterior eloszlását egy következő elemzés priorjaként használni? Több lehetőség is adott.[37] Ezek közül itt egy olyan lehetőséget mutatunk be, amelynek kapcsán elméleti mozzanatok mellett egy fontos technikai részlet is tisztázhatunk.

[37] pl. az itt bemutatott mellett lehet ugyanilyen célból metaanalízist is végezni

A példánkat folytatva: azt láttuk, hogy ahhoz, hogy prior ismereteket tudjunk a `dbeta()` eloszlással az elemzésbe bevonni, szükségünk van az a és a b paraméterekre. A flat prior esetén olyan paramétereket adtunk meg, amelyek 0 és 1 között minden értéknek ugyanolyan valószínűséget feltételeztek. Illetve (Tamás elképzelése alapján) a Betabuster szoftverrel határoztuk meg a paramétereket. De fontos látni, hogy ezeket a paramétereket szintén tudjuk becsülni a WinBUGS segítségével (nem csak a poszterior-eloszlást). Ahhoz, hogy ezt megvalósítsuk , mind az a, mind a b paraméterre egy-egy prior eloszlást kell megfogalmaznunk. Így egy újabb szintje lesz a modellünknek, a binomiális likelihoodhoz tartozó béta prior a és b paramétereinek is lesz egy-egy priorja. Amiket *hiperpriornak*[38] nevezünk.

[38] hyperprior

Nem konjugált priorok

Mivel a béta-eloszlás paraméterei nagyon változatos pozitív értékeket vehetnek fel,[39], illetve mivel nincsen elképzelésünk arról, hogy milyen tartományba eshetnek, olyan prior-eloszlást kell választanunk, amivel széles tartományban tudunk minden egyes lehetséges értékre azonos valószínűségeket leírni.[40] Ilyen priorokat számos eloszlással meg tudunk adni, azonban fontos azt megjegyezni, hogy a béta eloszlásnak nincsen konjugált priorja (1. táblázat). Így mivel nem konjugált priort kell használnunk az eddigi MC helyett, az elemzés során MCMC-szimulációt[41] fogunk használni, amivel kapcsolatban fontos szempontokat kell figyelembe venni. A modellünket a Lam-

[39] lásd Malacok ivararánya V.

[40] flat prior

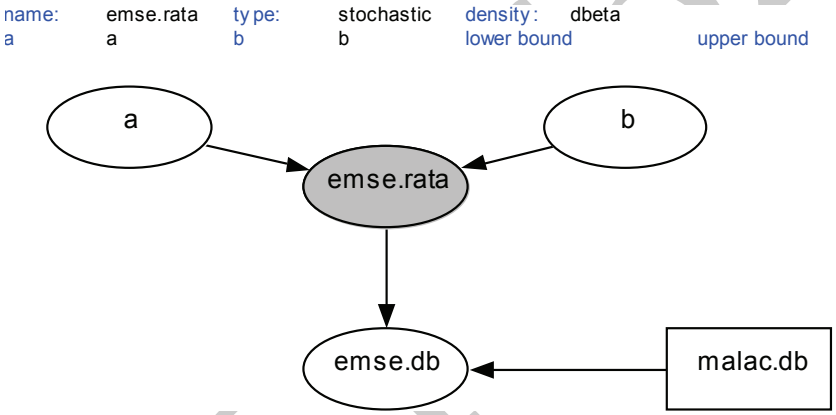
[41] *„Beware: MCMC sampling can be dangerous!"*

berson et al. (1988) adatokat felhasználva így írjuk fel:

```
model{
  emse.db ~ dbin(emse.rata, malac.db)
  emse.rata ~ dbeta(a, b)
  a ~ dunif(0, 1000)
  b ~ dunif(0, 1000)
}

list(
  emse.db=540, malac.db=1187
)
```

Itt a béta-eloszlás a és b paramétereire vonatkozóan *uniform*[42] prior eloszlást használunk. Ahogy a szkriptből látható, mind az a, mind a b paraméterre vonatkozóan az gondoljuk, hogy 0 és 1000 között bármely értéket ugyanolyan valószínűséggel vehetik fel. Azáltal, hogy ezekre a paraméterekre vonatkozóan bizonytalan, nem informatív priort határoztunk meg az `emse.rata` béta-prior is ilyen, nem informatív lesz. A modellt hiperpriorokkal a Doodle-szerkesztőben a 20. ábra mutatja.

[42] Az uniform eloszlásnak két paramétere van, az elsővel az eloszlás tartományának minimum, a másodikkal a maximum értékét határozhatjuk meg.



20. ábra. Modell hiperpriorokkal a doodle-szerkesztőben

Az eddigi modelljeink futtatásakor az iniciáló, kezdő értékekről nem beszéltünk, mivel az MC esetén annak nincsen nagy jelentősége. Azonban az MCMC-szimuláció alkalmazásakor fontos kérdés a láncok kezdőértékeinek megadása. A modellünk minden ismeretlen paraméterére meg kell adnunk kezdőértékeket, minden lánchoz külön külön. A példánkban három ismeretlen paraméterünk van, az `emse.rata`, az `a` és a `b`.

A kezdőértékeknek abból az eloszlásból kell származniuk, amit a paraméter becslése céljából meghatároztunk. Ennek megfelelően az `emse.rata` paraméter kezdőértéke 0 és 1 közé, `a` és `b` kezdőértéke 0 és 1000 közé kell hogy essen.

Ha egy lánccal fordítjuk[43] a modellünket, akkor mindegyik ismeretlen paraméterre egy-egy kezdőértéket kell megadnunk. Ha két lánccal[44] fordítjuk a modellt, akkor mindegyik ismeretlen paraméterhez kettő-kettő kezdőértéket kell megadnunk.

Egyszerűbb modellek esetén a `Specification tool` ablakban a *check model* → *load data* → *compile* lépések után generáltatunk kezdőértékeket a `gen inits` gomb megnyomásával. (8. ábra). A generált kezdőértékeket megnézhetjük a `Model` menü, `Save State` menüpontjának segítségével. A menüpontra kattintva lánconként egy-egy `State of Sampler` feliratú ablak jelenik meg, ami tartalmazza a kezdőértékeket. Pl. ilyen értékekkel:

```
list(
a = 242.1693146425156,
b = 139.6711967604566,
emse.rata = 0.6005419674775117)

list(
a = 970.0365420291371,
b = 404.1618837063023,
emse.rata = 0.7190258839915382)
```

Látható, hogy ugyanolyan listaformában tartalmazza a kezdőértékeket a szkript, mint amilyet a modellhez tartozó adatok megadásánál használunk.

Ha több, mint egy láncot futtatunk, akkor érdemes a nem ismert értékű paraméter eloszlásából olyan kezdőértékeket választani, amelyek elég távol vannak egymástól. Ahogy a szkriptben is látható, az egyik láncban *a* kezdőértéke 242.17, a másik láncban 970.04.[45] Olyan priorok esetén, amelyek nagyon „bizonytalanok" a WinBUGS esetenként olyan kezdőértékeket generál, amelyek használatával a szoftver összeomlik. Ilyen esetben újra kell paramétereznünk a modellünket, új kezdőértékeket kell generáltatni vagy megadni manuálisan.

A kezdőértékek manuális megadása a fenti szkriptben látható listák segítségével történik. Ilyenkor a *check model* → *load data* → *compile* lépések után a kezdőértékeket tartalmazó lista `list` szavát kijelöljük az egérrel és a `load inits` gombra kattintunk. Ha két láncot futtatunk, akkor a gomb első megnyomása után a `for chain` mezőben az 1 értéke 2-re vált, jelezve azt, hogy a gomb következő megnyomásával a második lánchoz tölthetünk be kezdőértékeket.

Lehetőség van arra is, hogy az automatikus és manuális kezdőérték megadást kombináljuk. Ami gyakorlatilag azt jelenti, hogy ha a `load inits` útján a modellünkben lévő nem ismert paramétereknek csak egy részére töltünk be kezdőértéket, akkor a többihez generálhatunk a `gen inits` gomb segítségével.

[43] compile

[44] `num of chains` mezőben 2-t írtunk

[45] Ennek a jelentőségét a konvergenciavizsgálat kapcsán látjuk majd.

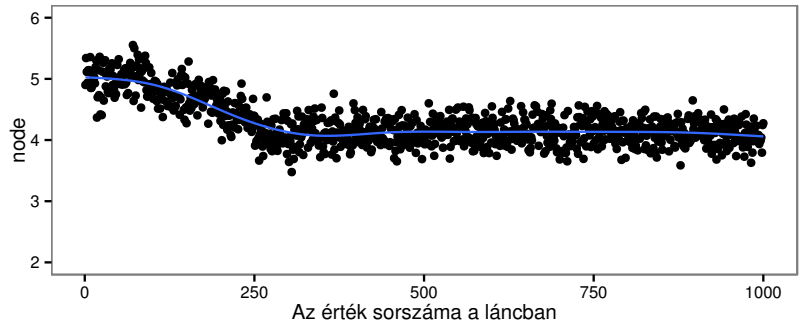
Akár generáltuk, akár manuálisan adtuk meg a kezdőértékeket, tanácsos eltárolni őket, amit leginkább úgy érdemes megtenni, hogy a szkriptünkbe illesztjük listaként.

Konvergencia

Mind az MC-, mind az MCMC-szimulációk esetén a poszterior eloszlásból vesz nagy számú mintát a WinBUGS a node-okra vonatkozó statisztikák számítása céljából. Míg az MC-szimulációk használatánál, vagyis a konjugált priorok esetében nem kell foglalkoznunk azzal, hogy a szimuláció eredményeként létrejött láncok a poszterior-eloszlásból származnak vagy sem, az MCMC esetén ez központi kérdés. Anélkül, hogy ennek a kérdésnek matematikai, számítástechnikai részleteire[46] kitérnénk, fontos annak bemutatása, hogyan ellenőrizhetjük, hogy a generált láncok, vagy azok egyes szakaszai valóban a poszterior-eloszlásból származnak, és így megbízhatóan használhatók fel statisztikai következtetések levonására.

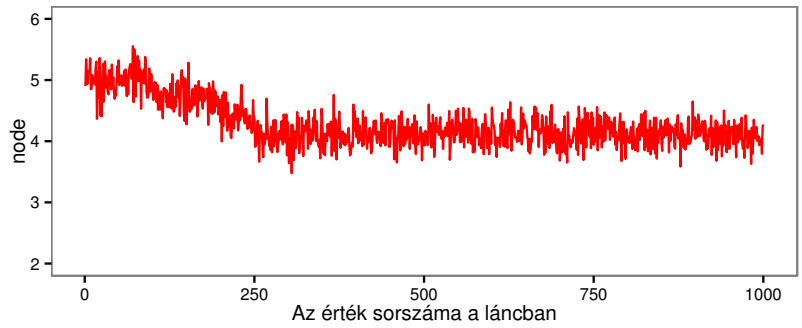
[46] lásd Lunn et al. (2012); Gelman et al. (2014)

Az MCMC-szimuláció során a láncokban értékek generálódnak, amely értékek (leginkább a lánc kezdeti szakaszaiban) jelentősen eltérhetnek a poszterior eloszlásra jellemző értékektől. A 21. ábrán látható egy lánc, amely 1000 mintavételből származó értékeket tartalmaz. A fekete pontok a lánc értékeit, a kék görbe az azokra illesztett nem-lineáris trendet mutatja. Ugyanezt a láncot láthatjuk a 22. ábrán, ahol az egymást követő értékeket egyenesek kötik össze, ugyanúgy ahogy a WinBUGS jeleníti meg (12. ábra).



21. ábra. MCMC-szimuláció értékei generálásuk sorrendjében, pontokkal jelölve, illetve a nem-lineáris trend

A láncok felhasználhatósága szempontjából több tulajdonságukat is meg kell vizsgálnunk. Az egyik alapvető vizsgálat során tulajdonképpen azt vizsgáljuk, hogy a lánc értékei a láncnak melyik szakaszában értik el a poszterior-eloszlást. Amikor a lánc elér egy stacionárius eloszlást, abban a szakaszában a poszterior eloszlásból származnak az értékei. *Konvergenciavizsgálat* során azt vizsgáljuk, hogy az MCMC-láncok elérik-e, és ha igen, akkor melyik szakaszukban a stacionárius

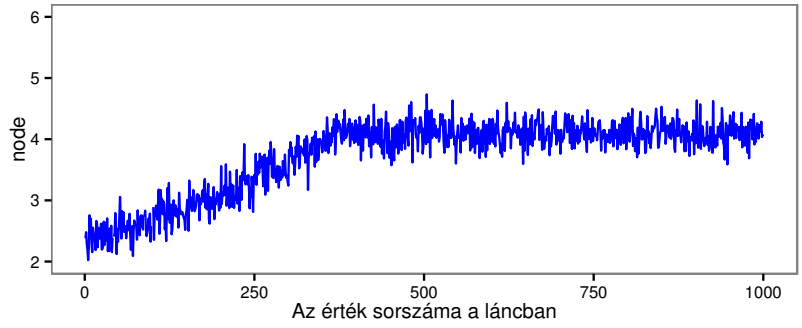


22. ábra. MCMC-szimuláció értékei generálásuk sorrendjében. Az egymást követő értékek egyenessel vannak összekötve

állapotot. Ezt vizsgálhatjuk a lánc értékeinek grafikus ábrázolása alapján szemmel, illetve különböző diagnosztikai mutatók felhasználásával kvantitatív módon.

A lánc stacionárius állapota előtti szakaszt *burn-in* szakasznak szokták nevezni, és azt nem helyes a statisztikák számításába bevonni, vagyis gyakorlatilag ki kell hagyni abból.

Láncok vizuális konvergenciavizsgálata A 21. és 22. ábrán látható, hogy körülbelül a lánc 250. eleméig trendjében eltérő a lánc az azutáni szakasztól. A lánc utolsó kétharmadára egy nagyjából vízszintes egyenest illeszthetünk. Ez utóbbi szakaszt stacionáriusnak tekinthetjük, vagyis ha a lánc 250. elem utáni értékeit használjuk a statisztikák számítására, akkor a poszterior eloszlásra vonatkozóan szerzünk ismereteket.



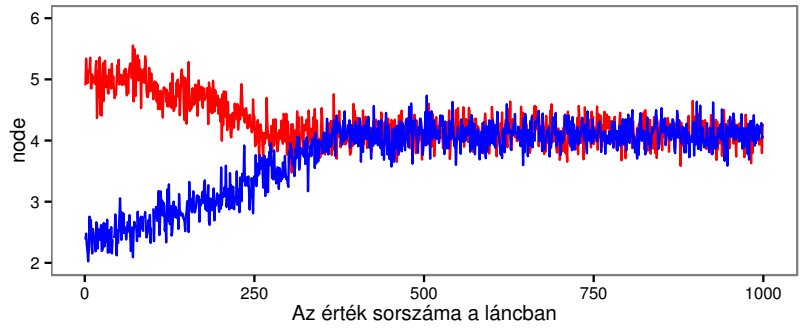
23. ábra. MCMC-szimuláció értékei generálásuk sorrendjében az előzőtől eltérő kezdőértéktől indítva

A 23. ábrán látható lánc ugyanabból a mintavételi eloszlásból származik, mint a 22. ábrán bemutatott, azonban a két lánc különböző kezdőértékekből[47] induló szimuláció eredménye. Ezen az ábrán az látható, hogy a stacionárius szakasz körülbelül a 350. értéktől kezdődik, vagyis 100 értékkel kevesebből számolhatók a szokásos statisztikák. A kezdőértékek megfelelő megválasztása azért is fontos, hogy minél rövidebb *burn-in* szakaszt kelljen kihagynunk az elemzésből és

[47] kezdőérték

minél hamarabb elérje a lánc a stacionárius állapotot.

A 21-23. ábrákon aránylag könnyű eldönteni, hogy mely szakasz tekinthető stacionáriusnak. Más esetekben ez vizuálisan nehezen eldönthető. Segíthet ebben, ha különböző kezdőértékű láncokat egy közös ábrán vizsgálunk. A 24. ábrán a 22. és a 23. ábrán bemutatott láncokat ábrázoltuk együtt.

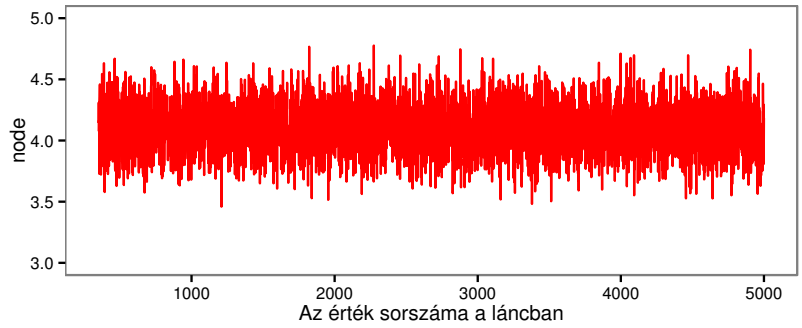


24. ábra. Két, ugyanabból a mintavételi eloszlásból származó lánc különböző kezdőértékekből indítva

A láncok egy ábrán való vizsgálatakor meggyőződhetünk arról, hogy a különböző kezdőértékekkel indított láncoknak van-e olyan szakasza, amit „kevertnek" tekinthetünk, vagyis nagymértékben átfed a két lánc között. A 24. ábrán látható, hogy a 350. mintától a két lánc nagymértékben átfedő. Ha egy lánc vizsgálata nem meggyőző abban a vonatkozásban, hogy a lánc hol válik stacionáriussá, akkor a két lánc együttes vizsgálata segíthet a döntésben.

A statisztikai következtetésekre alkalmas *burn-in* utáni stacionárius lánc hosszabb futás után úgy néz ki, mint mint egy „kövér, szőrös hernyó"[48] (25. ábra).

[48] Lunn et al. (2012)

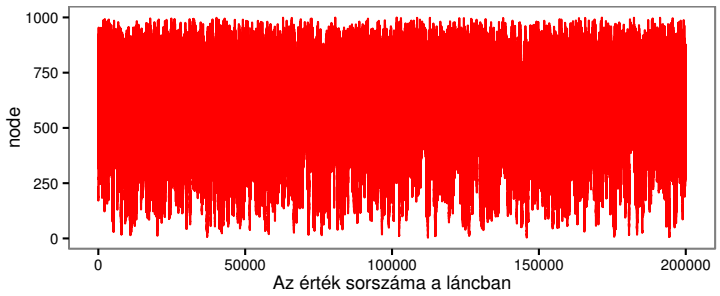


25. ábra. MCMC-lánc stacionárius állapotából vett minták rajzolata („kövér, szőrös hernyó")

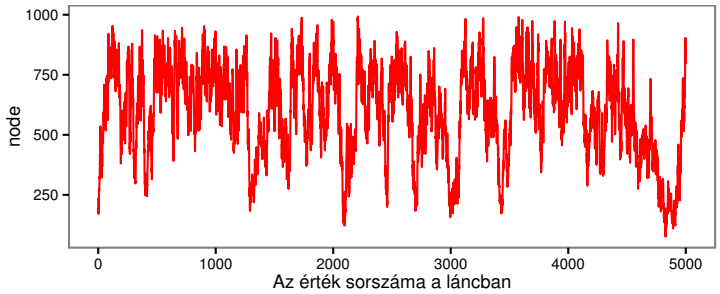
Azonban a vizuális konvergenciavizsgálat esetenként nagyon megtévesztő lehet. Ha a 26. ábrát nézzük, akkor úgy tűnhet, hogy egy nagy variabilitású, de stacionárius láncot látunk. Azonban fontos azt észrevenni, hogy amit látunk, az 200000 mintavételből létrejött lánc, vagyis a lánc kezdeti szakaszait vizsgálva lehet, hogy más ké-

pet kapunk a lánc használhatóságára vonatkozóan. Ha ugyanennek a láncnak csak az első 5000 elemét ábrázoljuk, akkor módosítanunk kell a 26. ábra alapján kialakult véleményünket (27. ábra). Ez a típusú, „kígyó”[49] alakot mutató lánc arra utal, hogy az értékek között valamilyen mértékű autokorreláció van, amit a korrekt elemzések céljából kezelni kell (ezt az autokorrelációról szóló szakaszban (31. lap) részletezzük).

[49] Lunn et al. (2012)



26. ábra. „Kövér, szőrös hernyónak” látszó, megtévesztő lánc



27. ábra. Konvergáló, de erős autokorrelációval terhelt lánc („kígyó”)

Láncok konvergenciájának vizsgálata diagnosztikai mutatók segítségével
Ahogy az előző példából is látszik, a konvergencia pusztán vizuális értékelése rejthet buktatókat. Ennek megfelelően számos olyan konvergenciavizsgálati módszert fejlesztettek, amelyek valamilyen statisztikán alapulva számszerűsíteni képesek a láncok, illetve azok egyes szakaszainak stacionárius voltát (Heidelberger & Welch, 1981, 1983; Geweke, 1992; Raftery & Lewis, 1992). Vannak módszerek, amelyek egyetlen, mások több lánc együttes együttes vizsgálatára alapozva segítik a konvergenciavizsgálatot.

A WinBUGS-ban egyetlen konvergenciavizsgálati módszer, a két láncra épülő *Brooks-Gelman-Rubin*[50] diagnosztika érhető el, aminek alapjait Gelman & Rubin (1992) mutatta be, azt pedig Brooks & Gelman (1998) módosította. Az eljárás lényege, hogy a láncok vagy azok egy szakaszának stacionárius állapotát úgy vizsgálja, hogy összeveti a láncok értékeinek variabilitását láncon belül és láncok között.

[50] BGR

A *Gelman-Rubin* módszer (Gelman & Rubin, 1992) lényege a következő: Két olyan láncból indulunk ki, amelyek mindegyikében n számú érték van.

Első lépésben mindkét láncból töröljük az első felét, vagyis csak a futás második feléből származó adatokat használjuk a továbbiakban. Ezeknek a csonkolt láncoknak az értékeiből kivesszük pl. a középső 80 percentilist.[51] Ennek a központi tartománynak a maximum értékéből kivonjuk a minimum értékét, ami megadja a tartomány szélességét. A két lánc szélességének az átlagát véve megkapjuk a W statisztikát, ami a *within*, vagyis a láncon belüli variabilitást számszerűsíti.

A következő lépésben a két csonkolt láncot, egyesítjük ebből is kiszámoljuk a középső 80 percentilis szélességét, ez lesz a *between*, vagyis a láncok közötti variabilitás mértéke, amit B jelöl.

E kettő variabilitás hányadosa $\hat{R} = B/W$, amit *potetial scale reduction factor (PSRF)* neveznek a szakirodalomban. Ha a kezdőértékek megfelelően távol[52] voltak a poszterior-eloszlástól, akkor $\hat{R} > 1$ kell, hogy legyen. A vizsgált láncszakaszok konvergáltnak tekinthetők, ha az $\hat{R} < 1.05$.

Brooks & Gelman (1998) több módosítást is bemutatott a Gelman & Rubin (1992) módszerre vonatkozóan, pl. annak többváltozós változatát fejlesztette ki. De emellett az $\hat{R}$ becslésére azt javasolta, hogy ne egy értékkel írjuk le a láncokat, hanem a láncok különböző szakaszaira számoljuk ki a B, a W és R értékeket, amiket aztán ábrázoljunk grafikusan. Így a mutatóra alapozott diagnosztikai eljárás vizuális értékelést tesz lehetővé. Ezt a módszert nevezzük BGR-nek. Az alapját képező számítási lépések a következők:

Míg a Gelman-Rubin módszer esetén csak a láncok második szakaszát, addig a BGR-nél a láncok teljes hosszát használjuk. Ehhez először meg kell határoznunk egy olyan tartományméretet (a), amire alapozva fogjuk kiszámolni a statisztikáinkat (B, W, R). A WinBUGS az a értékét úgy határozza meg, hogy minimálisan 100 legyen, vagy ha a lánc hosszának egy százada nagyobb 100-nál, akkor az lesz a értéke. Majd kiszámoljuk, hogy a láncban hányszor fér el az a hosszúságú szakasz, ennek jele Q. Következő lépésben kiszámoljuk, hogy a későbbiekben vizsgált láncszakaszoknak mi lesz a felső határa $1, ..., q \times a$, az összes q-ra[53] vonatkozóan.[54] A vizsgálandó szakaszok alsó határát úgy határozzuk meg, hogy a felső határnak a felét vesszük és hozzáadunk egyet.[55] A tartományok láncon belüli alsó és felső határának meghatározása után mindegyik ilyen tartományra kiszámoljuk a B, a W és R értékeket. Ezek tartományhoz tartozó értékeit a tartomány kezdő pontjának függvényében ábrázoljuk (28. ábra). A 28. ábráról több információ leolvasható, mint a Gelman-Rubin eljárás $\hat{R}$ értékéből. Az ábrán látható, hogy az $\hat{R}$-görbe

[51] Általánosítva ezt úgy mondhatjuk, hogy a lánc $100 \times (1 - \alpha)$ CR-jét vesszük. Az itt bemutatott példában $\alpha = 0.2$.

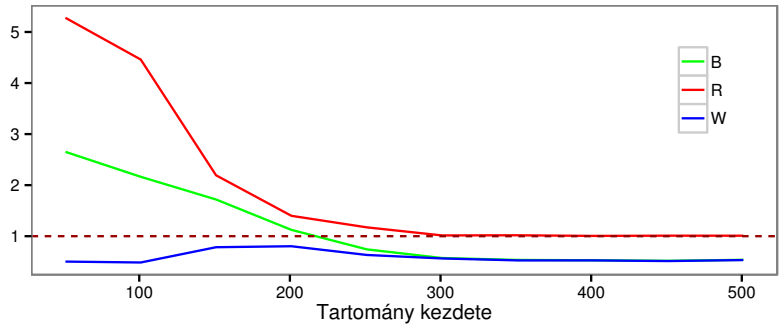
[52] overdispersed

[53] $q = 1, ..., Q$

[54] Ha a láncaink 1000 elemből állnak és $a = 100$, akkor $Q = 10$ és a vizsgált szakaszok felső határa: $1 \times 100 = 100$, $2 \times 100 = 200$, $3 \times 100 = 300$, $4 \times 100 = 400$, $5 \times 100 = 500$, $6 \times 100 = 600$, $7 \times 100 = 700$, $8 \times 100 = 800$, $9 \times 100 = 900$, $10 \times 100 = 1000$.

[55] $(q \times a/2) + 1$

hol éri el az 1 értéket. Míg a Gelman-Rubin eljárás alapján csak a lánc utolsó 500 értékét használhatjuk megbízható módon a statisztikai becsléseinkben, addig a 28. ábrán az $\hat{R}$-görbe azt jelzi, hogy a lánc már a 300. értékétől kezdve stacionárius.



28. ábra. BGR-diagnosztikai ábra a 24. ábrán bemutatott láncok adatai alapján

Az ábráról az is leolvasható emellett, hogy a lánc 300. elemétől a *B*- és *W*-görbék is olyan állapotba jutnak, amelyben nincsenek jelentősebb változások a lefutásukban. Ezt úgy szoktuk értelmezni, hogy a láncok stabilizálódtak az adott szakaszban.

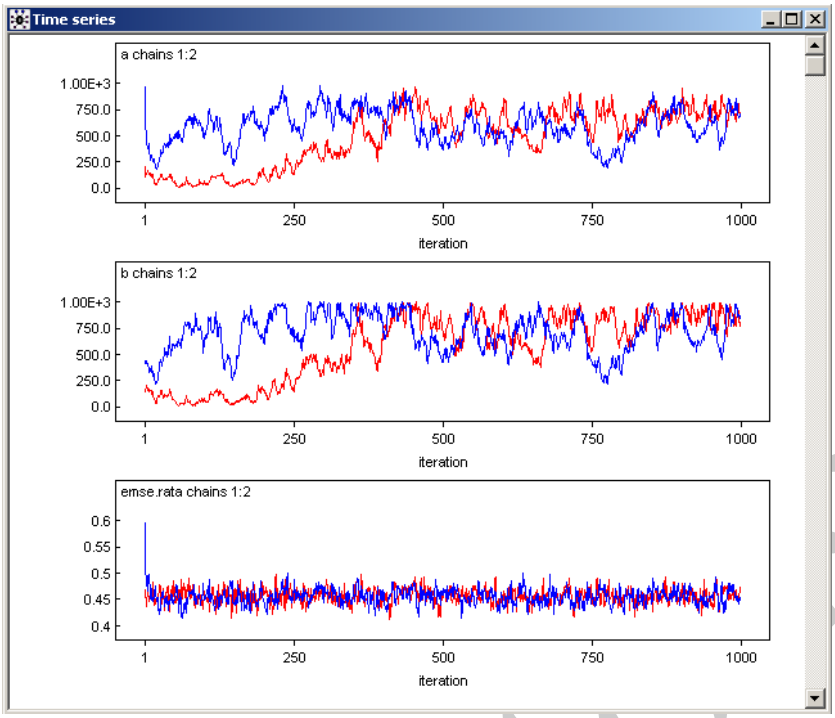
Fontos megjegyezni, hogy egyetlen diagnosztikai eljárás sem tökéletes, így (a biztonság kedvéért) a láncok konvergenciájának többféle szempontból való vizsgálata erősen ajánlható.

Miután a konvergenciával és vizsgálatával kapcsolatos fontosabb ismereteket vázoltuk, érdemes befejezni legutóbbi, nem konjugált priorokat tartalmazó példánkat (21. lap), úgy, hogy a futtatást konvergenciavizsgálattal is kiegészítjük. Az MCMC-szimulációt két lánccal végezzük, lefutása után pedig először nézzük meg,[56] hogy a láncok első 1000 eleme milyen lefutást mutat (29. ábra).

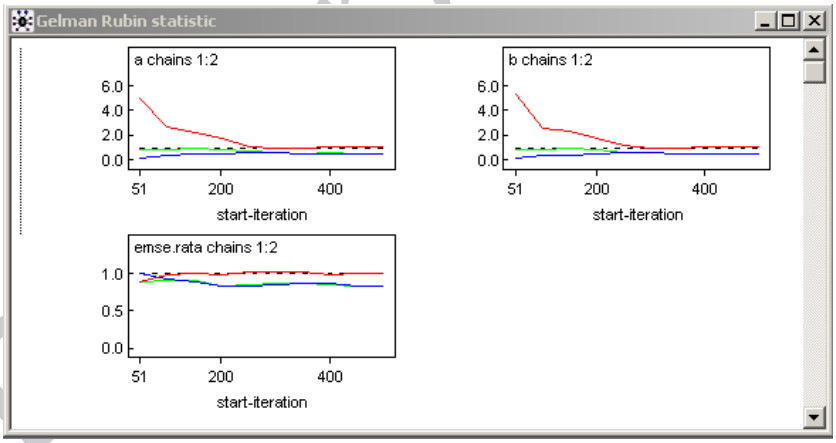
[56] `Sample Monitor Tool` ablak `history` gombjának segítségével

A 29. ábrán azt láthatjuk, hogy a hiperpriorok (`a`, `b`) láncai kisebb mértékben keverednek, míg a `emse.rata` láncai szépen keverednek. Ezek alapján gondolhatjuk azt, hogy `emse.rata` láncok hamar elérik a stacionárius állapotot, míg a másik kettő lassabban. De a vizuális konvergenciavizsgálatot ki kell egészítenünk a BGR-diagnosztikával. A WinBUGSban a `Sample Monitor Tool` ablak a `bgr diag` gombjának segítségével készíthetünk a 28. ábrához hasonló diagnosztikai grafikont a `chains` mezőkben meghatározott láncok, `beg` és `end` meghatározott szakaszára vonatkozóan.

A 30. ábrán látható, hogy a `emse.rata` láncai már a 100. elemüktől stacionárius állapotba kerülnek és stabilizálódnak. A két hiperparaméter esetén ez később, 300. elemtől alakul ki. A nem konvergáló szakaszokat ki kell hagynunk (burn-in), amikor a láncok értékei alapján statisztikákat számolunk. A `Sample Monitor Tool` ablak `beg` és



29. ábra. A nem konjugált példa MCMC-szimulációiból származó láncok első 1000 elemének görbéi. A legfölső és középső ábrán `a` és `b` hiperpriorok, az alsón pedig a `emse.rata` node látható. Utóbbin a két lánc keveredése egyértelmű, míg az előző kettőnél a láncok nem keverednek megfelelően, illetve „kígyó”-rajzolatot mutatnak, ami erős autokorrelációra utal.



30. ábra. A nem konjugált példa MCMC-szimulációiból származó láncok első 1000 elemének BGR-diagnosztikai görbéi.

`end` mezőiben adhatjuk meg, hogy a láncok melyik szakaszából kívánjuk a `stats` gomb segítségével kiszámoltatni az egyes node-okra vonatkozó statisztikákat. Mivel a láncok használható tartománya nem egyenlő széles, ezért két lépésben számíttatjuk ki a statisztikákat. Első lépésben az $101 - 1000$ értékek alapján a `emse.rata` node-ra vonatkozóan:

```
node       mean   sd       MC error  2.5%    median  97.5%   start  sample
emse.rata  0.456  0.01427  5.627E-4  0.4277  0.4559  0.4844  101    1800
```

Majd a $301 - 1000$ tartományra a `a` és `b` node-okra:

```
node  mean   sd     MC error  2.5%   median  97.5%  start  sample
a     621.3  155.8  18.04     298.8  642.2   872.8  301    1400
b     737.5  175.7  20.24     366.8  766.8   987.7  301    1400
```

Mivel két láncot szimuláltunk, mindegyik node-ra 1800, illetve 1400 értékből számítódtak a statisztikák.

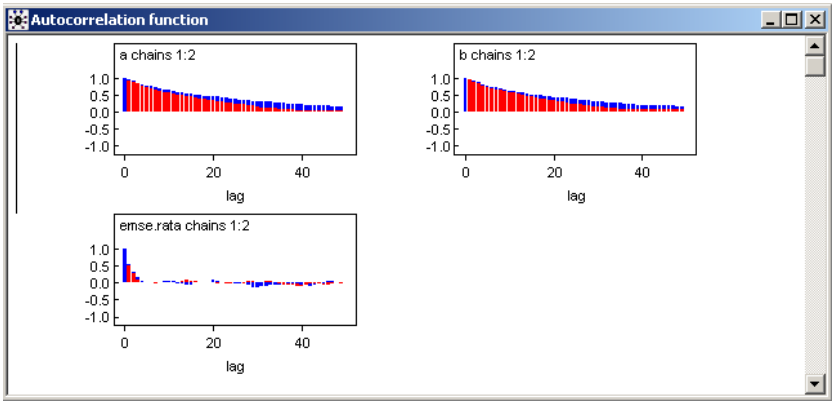
Az eredménytáblázatok alapján megkaptuk azokat a becsléseket, amelyekre kíváncsiak voltunk, vagyis, hogy Lamberson et al. (1988). adatai alapján milyen poszterior `a` és `b` értékeket használhatnánk egy következő elemzésben a `beta(a,b)` prior eloszlásunk meghatározásánál. Azonban mielőtt ezeket az eredményeket felhasználnánk, vissza kell térnünk a 29. ábrához, amelyen az `a` és `b` node-ok láncai „kígyó”-rajzolatot mutatnak, amelyről a 27. ábra kapcsán azt mondtuk, hogy autokorrelációra utal.

Autokorreláció

Elméletileg az MCMC-szimuláció során a láncok egymást közvetlenül követő elemei (láncszemei) nem függetlenek egymástól, de minden elem független a tőle legalább kettő elemnyi távolságban lévő elemektől. Azonban mégis előfordul, hogy a láncban nem közvetlenül egymás mellett álló elemek értéke nagyon hasonló, azaz korrelál. Mégpedig autokorrelál, mivel a láncon belüli egymástól való távolság függvényében hasonlítanak egymásra a lánc elemeinek értékei. Azok a láncok, amelyek autokorrelálnak, lassabb szimulációt eredményeznek.

Az MCMC-láncok használhatósági vizsgálata során a konvergenciavizsgálat mellett az autokorreláció mértékének értékelése is fontos. A 31. ábrán látható a WinBUGS autokorreláció vizsgálatot szolgáló ábrái a láncainkra vonatkozóan, amiket a `Sample Monitor Tool` ablak `auto cor` gombjának lenyomásával hoztunk létre.

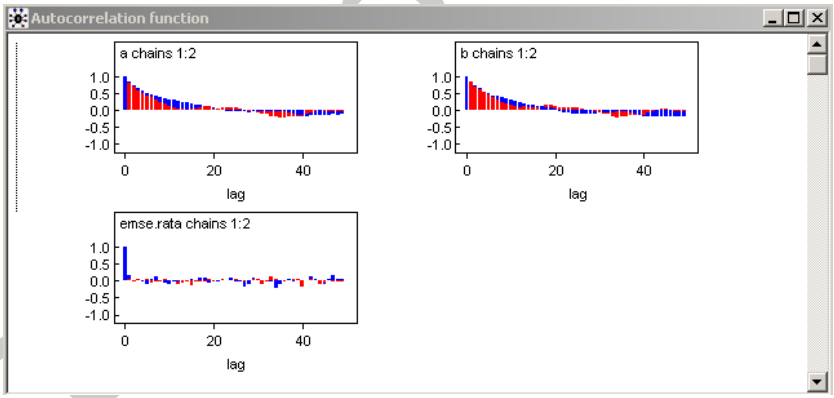
Az ábrán a két szín (piros, kék) a két láncot jelöli. Az oszlopok magassága a láncban adott távolságban (`lag`) lévő értékek közötti korrelációt jelzi. Az első oszlop a nulladik szomszédra, vagyis a lánc



31. ábra. A nem konjugált példa MCMC-szimulációiból származó láncok első 1000 elemének autokorreláció-diagnosztikai grafikonjai.

elemére önmagára vonatkozik és természetesen az értéke mindig 1. A következő oszlop minden elem első szomszédjával való korrelációját jelzi. Az azt követő oszlop minden elem második szomszédjával való korrelációját jelzi, és így tovább. A WinBUGS az ötvenedik szomszédig vizsgálja az autokorrelációt. Ezek szerint a `emse.rata` node-ban körülbelül a harmadik szomszédig van számottevő korreláció a lánc értékei között, azonban a `b` és `a` node-oknál, ahogy a 29. ábra alapján várható volt, nagy távolságban is jelentősen korrelálnak a lánc értékei.

Az autokorreláció csökkentésére a WinBUGS-ban az ún. *ritkítás (thining)* funkciója szolgál. Ennek lényege, hogy a láncokból nem minden elemet használunk, hanem csak minden valahányadik elemet. Azt javasolják, hogy a ritkítás során az autokorrelációs vizsgálatban még nem elhanyagolható mértékű korrelációt mutató legtávolabbi szomszéd sorszámát használjuk. A `emse.rata` node esetén a 31. ábra alapján ez az érték a 3.



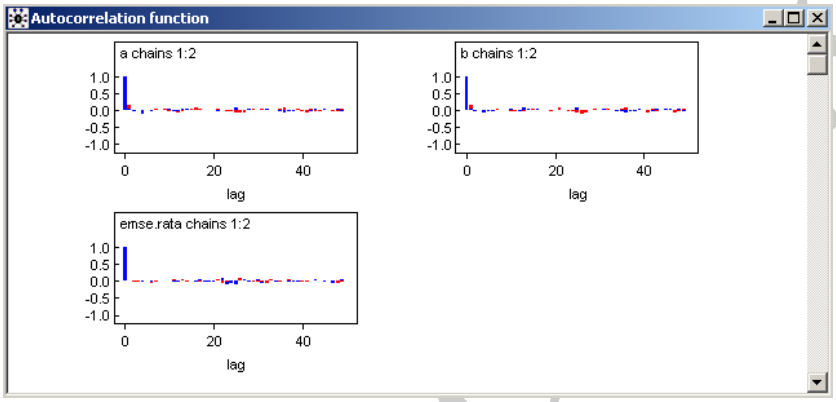
32. ábra. A nem konjugált példa MCMC-szimulációiból származó láncok első 1000 elemének autokorreláció-diagnosztikai grafikonjai *thin* = 3 beállítással.

Ha a `Sample Monitor Tool` ablak `thin` mezőjébe az 1 helyére 3-at írunk és így nézzük meg az autokorrelációs grafikonokat (32. ábra), akkor láthatjuk, hogy a `emse.rata` node esetén eltűnt az autokorreláció, a `a` és `b` node-oknál pedig csökkent a még nem elhanyagol-

ható szomszédossági mérték. Ezen a ponton felmerülhet bennünk a kérdés, hogy a ritkítás hogyan befolyásolja az ily módon kevesebb elemből álló láncokra alapozott statisztikákat.

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
a	621.5	154.7	21.16	305.5	643.2	872.8	301	466
b	736.5	177.2	24.56	357.2	760.0	986.2	301	466
emse.rata	0.4563	0.01406	5.31E-4	0.4281	0.4563	0.4852	101	600

Az eredménytáblázatból látszik, hogy a becslések nem változtak érdemben, de bizonytalanságra utaló mértékek (`sd`, `MC error`) megnőttek, ami természetes következménye annak, hogy a korábbi 1400 és 1800 elemből álló láncok helyett 466 és 600 elemű láncok alapján számítódtak a statisztikák.



33. ábra. A nem konjugált példa MCMC-szimulációiból származó láncok első 1000 elemének autokorreláció-diagnosztikai grafikonjai *thin* = 150 futtatással.

Ahhoz, hogy a `a` és `b` node-ok láncaiban is optimális szintre tudjuk lecsökkenteni az autokorrelációt, sokkal nagyobb ritkítást kellene végeznünk a láncokban, azonban 50 − 100-as `thin` értékkel alig marad valami a láncainkból. Ezért ilyenkor új MCMC-szimulációkra van szükségünk, amiket úgy futtatunk, hogy a WinBUGS ritkábban vesz mintát a szimulált értékek közül.

Ezt úgy tudjuk megtenni, hogy az `Update Tool` ablakban még az `update` gomb megnyomása előtt átírjuk a `thin` mezőben az 1-et pl. a mi esetünkbe 150-re. Így a szimuláció során csak minden 150. érték tárolódik el a láncunkban, úgy, hogy annak a szimuláció végén a hossza az `updates` mezőben megadott érték marad.

A 150-es ritkítással futtatott szimulációból származó láncok autokorrelációs grafikonjai a 33. ábrán láthatók. Az ábrán látható, hogy a ritkítás eredményeként az `a` és `b` node-ok autokorrelációja is elfogadható szintre csökkent.

A 150-es ritkítású láncok elemeinek értékeiből számított statisztikáknál azt kell észrevennünk, hogy az `a` és `b` node-okra vonatkozó becslések jelentősen módosultak.

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
a	564.8	201.0	5.756	138.7	589.2	879.9	301	1400
b	670.6	231.4	6.64	163.1	713.9	985.3	301	1400
emse.rata	0.4557	0.01467	3.311E-4	0.4282	0.4553	0.487	101	1800

Vagyis az erős autokorreláció nem csak lassítja a láncok létrejöttét, de jelentős hatással lehet a becsléseinkre vonatkozóan is.

Poszteriorból prior

Az előző szakaszban az MCMC-szimulációkból származó láncok felhasználhatóságának vizsgálata mellett a Lamberson et al. (1988) által közölt adatok alapján meghatároztuk az emsék alombeli előfordulásának modellezésében használt `beta(a,b)` prior paramétereit. Annak az elemzésnek egyebek mellett ezek voltak eredményei, a poszterior-eloszlásokból.

De ugyanezen eredmények a következő elemzéseinkben a `beta(a,b)` prior paraméterei lesznek. Egy egyszerűbb elemzésben azt vizsgáljuk, hogy Lamberson et al. (1988) által közölteknek a Bocian et al. (2012) adataival való kombinációja milyen változást jelent az emsék születési valószínűségének vonatkozásában.

```
model{
  emse.db ~ dbin(emse.rata, malac.db)
  emse.rata ~ dbeta(564.8, 670.6)
}

list(
  emse.db=73, malac.db=133
)
```

A futtatás eredménye:

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
emse.rata	0.4663	0.01359	1.376E-4	0.4396	0.4663	0.4932	1	10000

A MC-szimulációk alapján az emsék születési valószínűsége 47%-nak adódik. Bocian et al. (2012) adatainak flat priorral való elemzése 55%-os, Tamás priorjával való elemzése pedig 47%-os emse születési valószínűséget mutatott.

Ezek az eredmények jól szemléltetik a bayesi elemzéseknek azt a tulajdonságát, hogy a „bizonytalan”[57] priorok esetén a poszterior-

[57] vague, non informative, flat

eloszlás az adatok által dominált, míg határozott prior esetén a priornak jelentős befolyása lehet a poszterior eloszlásra, különösen akkor, ha kevés „kemény” információt tartalmaz a likelihood. Ezért gyakran hangsúlyozzák a bayesi modellezéssel kapcsolatban, hogy nagyon fontos a prior megfelelő megválasztása, illetve a modell priorra vonatkozó szenzitivitás-vizsgálata. Ennek során különböző, vagy különbözően paraméterezett priorokat próbálnak ki, és az eredmények összevetésével azt vizsgálják, hogy van-e jelentős hatása az eltérő prioroknak, és ha igen, akkor melyik az, amelyik alkalmazása leginkább megindokolható.

Malacok ivararánya VII.

Az emsék részarányára vonatkozóan eddig csak olyan adataink voltak, amelyek több alom adatait összesítették, vagyis nem tudjuk, hogy azokban a vizsgálatokban, amelyekből ezek az adatok származnak az egyes almokban hány malac volt, és azok közül hány volt emse.

Azonban Tamás (visszatérve a szabadságáról) a telepén született almok közül véletlenszerűen kigyűjtött huszonkettőre vonatkozó ivararány-adatokat. Ezek felhasználása a BUGS-szkriptben eddig nem használt megoldást igényel:

```
model{
  for(i in 1:N){
    emse.db[i] ~ dbin(emse.rata, alommeret[i])
  }
  emse.rata ~ dbeta(564.8, 670.6)
}

# Tamás adatai
list(
   N=22,
   emse.db = c(5, 11, 3, 11, 8, 7, 6, 10, 7, 6, 7, 9, 7, 8, 6, 3, 7, 7, 7, 7, 5, 9,
   10, 7, 4, 5, 7),
   alommeret = c(14, 18, 14, 15, 16, 13, 15, 20, 18, 8, 14, 16, 15, 16, 14, 12, 17,
   15, 15, 16, 13, 15, 13, 15, 10, 15, 17)
)
```

Ahogy látható, a modellben van egy `for`-ciklus. A BUGS-nyelvben erre azért van szükség mert több alomra külön-külön rendelkezünk adatokkal, és azokat nem összesítve szeretnénk elemezni. A ciklus 1-től `N`-ig generál `i`-t. A ciklus minden lépésében adott `i`-nek megfelelő sorszámú elemet vesz ki a WinBUGS az `emse.db` és `alom.meret` vektorokból. A vektorok egyes elemeire való hivatkozás szintaxisa a

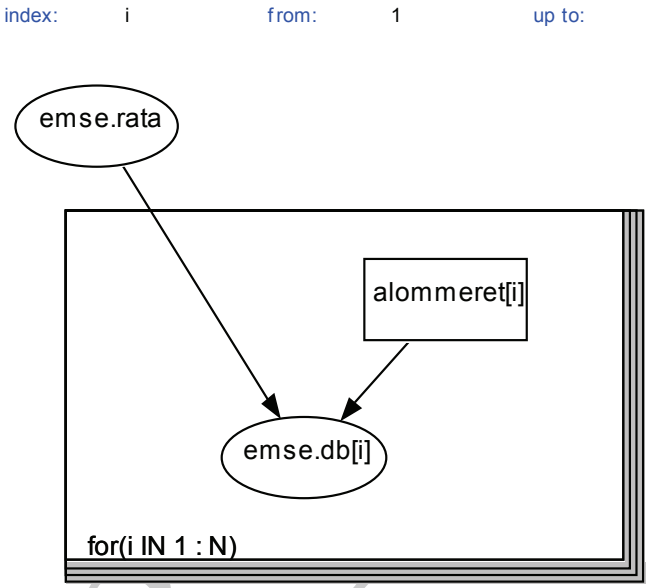
vektor neve után írt `[]`, amibe beírjuk az indexet. A BUGS-ban a ciklusok használata során nagyon fontos, hogy ha a ciklus hosszát nem egy számként (`for(i in 1:22)`), hanem változóként (`for(i in 1:N)`) adjuk meg, akkor a változónak értéket kell adni, vagy a modellben (`N<-22`), vagy az adatlistában (`N=22`).

Az almonkénti adatok alapján az összes alomra becslünk egy *közös* emse születési valószínűséget, a `emse.rata`-t, amiknek közös priorjának paramétereit a hiperprioros példából vettük.

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
emse.rata	0.4606	0.01256	1.367E-4	0.4358	0.4606	0.4855	1	10000

Ezek alapján úgy látszik, hogy az emsék születésének valószínűsége 46%.

Ha a Doodle-szerkesztőben szeretnénk a modellünket megszerkeszteni, akkor egy eddig nem használt, a `for` ciklust megjelenítő grafikus elemet kell elhelyeznünk a szerkesztőben (34. ábra).



34. ábra. Doodle-szerkesztő `for` ciklussal, közös emse rátával

A ciklust jelző elemet úgy tudjuk létrehozni, hogy a `Ctrl` gomb lenyomása mellett kattintunk a Doodle-szerkesztő felületére az egérrel. Ennek a grafikus objektumnak a paraméterezése során meg kell adnunk a ciklusban a vektorok elemeire való hivatkozásra használt index nevét (`index`, illetve a ciklus tartományát (`from` és `up to`).

Az a node, amit a ciklus-téglalapra helyezünk a szerkesztőben, a generált kódban a cikluson belül lesz. Azonban figyelni kell arra, hogy ahhoz, hogy a node indexelve is legyen, a nevének vége a indexelés jele: a `[]` kell, hogy legyen, természetesen beleírva az index nevét is (esetünkben `[i]`).

Az eddigi példákban az adatainkat vagy a modellbe írtuk ,vagy külön listaként adtuk meg, de a BUGS-ban más adatszerkezetek is használhatók, pl. Tamás adatait megadhatjuk mátrixként is:

```
list(N=22)

emse.db[]  alom.meret[]
    5          14
   11          18
    3          14
   11          15
    8          16
    7          13
    6          15
   10          20
    7          18
    6           8
    7          14
    9          16
    7          15
    8          16
    6          14
    3          12
    7          17
    7          15
    7          15
    7          16
    5          13
    9          15
   10          13
    7          15
    4          10
    5          15
    7          17
END
```

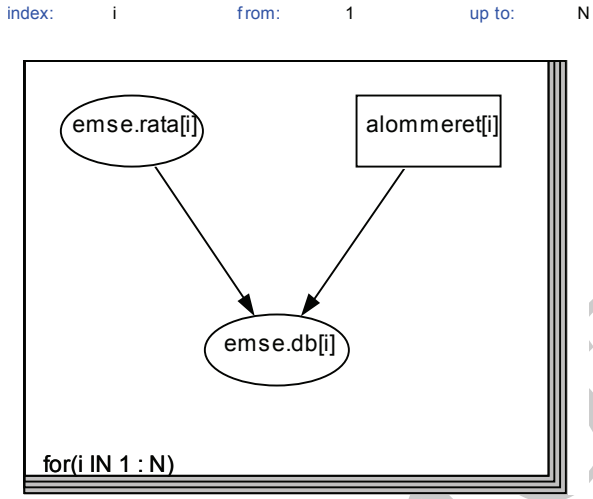
Ebben az esetben a mátix oszlopait szóközökkel választjuk el, elnevezésük során pedig a nevük végére illesztjük a `[]` jelet. A mátrix utolsó sora utáni sorba az `END` szót kell írnunk. A mátrix betöltése ugyanúgy történik a `Specification Tool` ablakban a `load data` gombbal, mint a listák esetében, annyi különbséggel, hogy itt nem a `list` szót jelöljük ki a betöltés előtt, hanem a mátrix első oszlopának a nevét. Ha több adattároló objektumunk is van a szkriptünkben, akkor azokat egymás után tölthetjük be. Tamás adatainak mátrixos példájában pl. az `N=22` tartalmú lista betöltése után betöltjük a mátrixot is.

A 34. ábrán és a hozzá tartozó szkriptben az `emse.rata` node a cikluson kívül helyezkedik el, ami azt eredményezi, hogy az összes alomra egy *közös* emse-születési valószínűséget becsülhetünk vele. Ha ezt a node-ot is betesszük a ciklusba, akkor minden egyes alomra

külön-külön becsülhetünk emse-ráta valószínűséget (35. ábra).

```
model{
  for(i in 1:N){
    emse.db[i] ~ dbin(emse.rata[i], alom.meret[i])
    emse.rata[i] ~ dbeta(564.8, 670.6)
  }
}
```

A modellben látható, hogy az `emse.rata` is egy vektor lesz, aminek minden eleme ugyanazt a `dbeta(564.8, 670.6)` priort kapja. Az

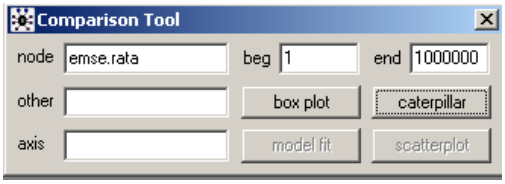


35. ábra. Doodle-szerkesztő `for` ciklussal almonkénti emse rátával

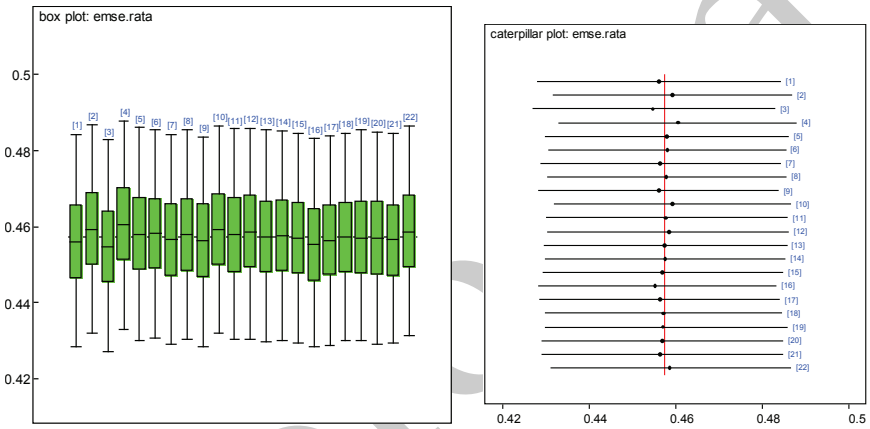
MC-szimulációkból származó láncok statisztikáját a szokásos módon a `Sample Monitor Tool` ablak `stats` gombjával számíttathatjuk ki, azonban figyeljünk arra, hogy itt a node neve a `emse.rata` és nem `emse.rata[]` vagy `emse.rata[i]` lesz. Az MC-szimulációk eredményeit ebben az esetben így írja ki a WinBUGS. Látható, hogy minden alomra a neki megfelelő indexszel egy-egy sort tartalmaz a táblázat.

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
emse.rata[1]	0.4561	0.01426	1.346E-4	0.4282	0.4558	0.4842	1	10000
emse.rata[2]	0.4593	0.01399	1.513E-4	0.4319	0.4594	0.4868	1	10000
emse.rata[3]	0.4547	0.01416	1.333E-4	0.427	0.4546	0.4828	1	10000
emse.rata[4]	0.4606	0.01409	1.514E-4	0.433	0.4605	0.4879	1	10000
emse.rata[5]	0.458	0.01421	1.43E-4	0.4299	0.458	0.4861	1	10000
emse.rata[6]	0.4581	0.01396	1.303E-4	0.4307	0.4581	0.4854	1	10000
emse.rata[7]	0.4564	0.01411	1.442E-4	0.4289	0.4564	0.4842	1	10000
emse.rata[8]	0.4578	0.01418	1.304E-4	0.4304	0.4577	0.4855	1	10000
emse.rata[9]	0.4562	0.01423	1.489E-4	0.4283	0.4561	0.4837	1	10000
emse.rata[10]	0.4592	0.01404	1.4E-4	0.432	0.459	0.4864	1	10000
emse.rata[11]	0.4577	0.01418	1.439E-4	0.4303	0.4577	0.4857	1	10000
emse.rata[12]	0.4585	0.01411	1.177E-4	0.4305	0.4585	0.4859	1	10000
emse.rata[13]	0.4573	0.01412	1.386E-4	0.4296	0.4572	0.4856	1	10000
emse.rata[14]	0.4576	0.01417	1.43E-4	0.43	0.4576	0.4853	1	10000
emse.rata[15]	0.457	0.01404	1.537E-4	0.4294	0.4571	0.4846	1	10000
emse.rata[16]	0.4552	0.01401	1.337E-4	0.4283	0.4552	0.4831	1	10000
emse.rata[17]	0.4564	0.01401	1.328E-4	0.4287	0.4565	0.484	1	10000
emse.rata[18]	0.4571	0.01401	1.333E-4	0.43	0.4571	0.4845	1	10000
emse.rata[19]	0.4571	0.01427	1.171E-4	0.43	0.4567	0.4856	1	10000
emse.rata[20]	0.4569	0.0141	1.514E-4	0.4292	0.4568	0.4848	1	10000
emse.rata[21]	0.4565	0.01409	1.392E-4	0.4292	0.4564	0.4846	1	10000
emse.rata[22]	0.4586	0.01407	1.489E-4	0.4313	0.4586	0.4866	1	10000

A WinBUGS-ban lehetőség van arra, hogy az ilyen megfigyelt vektor node-okat grafikusan is ábrázoljuk, a vektor egyes elemeinek összehasonlítása végett. Ezt az `Inference` menü `Compare...` menüpontjának segítségével érhetjük el a `Comparision Tool` ablakon keresztül. A node-vektor elemei ennek az eszköznek a segítségével boxplot vagy caterpillar ábrán hasonlíthatók össze (37. ábra).



36. ábra. `Comparision Tool` ablak, ami egy vektor node elemeinek grafikus összehasonlítását segítő felület



37. ábra. Node elemeinek grafikus ábrázolása boxplot (balra) és caterpillar (jobbra) ábrán a `Comparision Tool` ablak beállításai alapján

R2WinBUGS

Az eddigi WinBUGS-os tapasztalatok alapján is látható, hogy ez az eszköz nagyon rugalmas modellezési környezetet biztosít. Ezzel együtt számos korlátja is van, ilyen pl. a körülményes adatbevitel. Hogyha nagyszámú kezdőértékre van szükségünk, ráadásul több lánchoz is, akkor azok bevitele is körülményes. Habár a WinBUGS-ban vannak grafikai funkciók, az előállított ábrák hagynak kívánnivalót maguk után.

Mivel az R-környezet(R Core Team, 2015) minden tekintetben rugalmas statisztikai, adatbányászati, grafikai környezet, kézenfekvő, hogy az abban létrehozott, előkészített adatokat közvetlenül adhassuk át a WinBUGS-nak és az ottani szimulációk eredményeit további feldolgozás, grafikus ábrázolás céljából az R-környezetbe beolvassuk. Számos R-csomag biztosítja ezeket a lehetőségeket, itt az `R2WinBUGS`-ot mutatjuk be röviden (Sturtz et al., 2005). A csomag `bugs()` függvénye a megadott modell, adatok, kezdőértékek alapján a WinBUGS-ban lefuttatja a szimulációkat, az eredményeket pedig beolvassa az

R-környezetbe. A legelső, priorral már rendelkező példánkon (15. lap) keresztül tekintsük át a modellfuttatásnak ezt a módját.

```
1  # R
2  library(R2WinBUGS)
3
4  # modell leírása
5  model = function(){
6    emse.db ~ dbin(emse.rata, alom.meret)
7    emse.rata ~ dbeta(a, b)
8  }
9
10 # adatok megadása
11 emse.db = 540
12 alom.meret = 1187
13 a = 1; b = 1
14 adatok = list('emse.db', 'alom.meret', 'a', 'b')
15
16 # kezdeti értékek megadása
17 inits = function(){
18   list(emse.rata = rbeta(1, a, b))
19 }
20
21 # a vizsgálandó node-ok meghatározása
22 nodok = c('emse.rata')
23
24 # modell kiírása a munkakönyvtárba
25 setwd('C:/bugswd')
26 write.model(model, 'model.bugs')
27
28 # szimulációk futtatása a WinBUGS meghívásával
29 results = bugs(
30   data = adatok,
31   inits = inits,
32   parameters.to.save = nodok,
33   model.file = 'model.bugs',
34   bugs.directory='C:/Program Files/WinBUGS14/'
35   working.directory = getwd(),
36   n.chains = 1,
37   n.iter = 10000,
38   n.burnin = 0,
39   n.thin = 1,
40   debug = F,
41   useWINE = F)
```

Az R-szkript értelmezése Az 5-8. sorban leírjuk a modellünket. Ezt az R-ben függvény-definiálásként kell megtennünk, ügyelve arra, hogy a `{}` között nem R hanem BUGS-nyelven kell megírni a kódot. A 10-13. sorban az R-nyelv szabályai szerint értékeket adunk objektumoknak (`emse.db`, `alom.meret`, `a`, `b`). A 14. sorban létrehozunk egy `adatok` nevű listát, amiben az előző objektumok neveit felsoroljuk. Fontos észrevenni, hogy az objektumneveket idézőjelek közé kell írni. A 17-19. sorban a kezdeti értékekhez írunk fel egy függvényt a `model`-nél látottakhoz hasonlóan. Azonban itt a `{}` közé nem BUGS-nyelven, hanem R-nyelven hozunk létre egy listát, ami a kezdeti értékeket tartalmazza. Ebben a listában az egyes node-oknak megadhatunk konkrét értékeket, de ahogy a példában is látszik, a node priorjának eloszlásából véletlen értéket is vehetünk. Fontos látni, hogy az R és a BUGS szintaxisában különbség van: míg az R-ben `rbeta()` a függvény és három paramétere van, addig a BUGS-ban `dbeta()` és két paramétere van. A WinBUGS-ban a `Sample Monitor Tool` ablakban adjuk meg egyenként, hogy mely node-okról szeretnénk poszterior-információkhoz jutni, a `bugs()` függvény esetében egy vektorban adhatjuk meg a node-ok nevét (22. sor), itt is " között. A 25-26. sorban megadjuk a munkakönyvtárat, illetve modellünket kiírjuk ASCII-fájlként (`model.bug`) a `write.model()`-függvénnyel. A WinBUGS ezt a fájlt fogja beolvasni mint modellt a szimulációk futtatása előtt. A 29-41. sorok a `bugs()`-függvény paraméterezését mutatják be. A korábban létrehozott függvényeket, listákat és vektorokat a megfelelő argumentum értékeiként adjuk meg. Mivel korábban megadtuk a munkakönyvtárat, a `working.directory` értékadásánál a `getwd()`-függvényt használhatjuk, és nem kell újra beírnunk az utat. Egy láncot futtatunk (`n.chains=1`), amiben 10000 értéket generálunk (`n.iter=10000`). Mivel konjugált priort használ a modellünk, MC-szimulációt fog végezni a WinBUGS, így nincs szükségünk burn-in-re (`n.burnin=0`) és a ritkítással sem kell foglalkoznunk (`n.thin=1`). A `bugs()`-függvény futtatása alatt a WinBUGS elindul, és látjuk is a futás állapotait a képernyőn. Ha a `debug` argumentumnak `FALSE` értéket adunk, akkor a szimuláció végével a WinBUGS bezáródik. Ha `TRUE` értéket kap, akkor a futás végeztével nem zárul be, így lehetőségünk van a korábban ismertetett láncvizsgálatokat végrehajtani. Ha MS Windows-on futtatjuk a szkriptünket, akkor a `useWINE` argumentumnak `FALSE` értéket kell adnunk, ha Linuxon Wine használatával, akkor `TRUE` értéket.

A WinBUGS szimulációk eredménye egy `bugs` objektumként jelenik meg az R-ben, amiből az alábbi vektorokat, táblázatokat és listákat olvashatjuk ki:

```
 [1] "n.chains"        "n.iter"          "n.burnin"        "n.thin"
 [5] "n.keep"          "n.sims"          "sims.array"      "sims.list"
 [9] "sims.matrix"     "summary"         "mean"            "sd"
[13] "median"          "root.short"      "long.short"      "dimension.short"
[17] "indexes.short"   "last.values"     "isDIC"           "DICbyR"
[21] "pD"              "DIC"             "model.file"      "program"
```

Ahogy a nevekből látható, a `bugs` objektum nem csak az összefoglaló statisztikákat tartalmazza, hanem minden olyan adatot, statisztikát, amely a szimuláció és feldolgozás során létrejött, így pl. a láncok nyers értékeit is (`sims.matrix`). Ez lehetővé teszi, hogy az R által nyújtott széles eszköztárból használjunk függvényeket a `bugs` objektum további feldolgozására.

Az eredmények összefoglalásának kiíratásánál érdemes meghatározni a kerekítés mértékét:

```
1 print(results, digits.summary = 4)

Inference for Bugs model at "model.bug", fit using WinBUGS,
 1 chains, each with 10000 iterations (first 0 discarded)
 n.sims = 10000 iterations saved
            mean     sd   2.5%    25%    50%    75%   97.5%
emse.rata 0.4553 0.0145 0.4267 0.4457 0.4552 0.4649  0.4838
deviance  8.5284 1.4148 7.5240 7.6220 7.9600 8.8712 12.5902

DIC info (using the rule, pD = Dbar-Dhat)
pD = 1.0 and DIC = 9.5
DIC is an estimate of expected predictive error (lower deviance is better).
```

rjags

A Martyn Plummer által fejlesztett, a WinBUGS-hoz hasonló szimulációk futtatására alkalmas eszköz a *Just Another Gibbs Sampler* (JAGS). A programot C++ programozási nyelven fejlesztették abból a célból, hogy egy nyílt forráskódú, platformfüggetlen bayesi modellezést lehetővé tevő alkalmazás legyen elérhető bárki számára. A JAGS parancssoros[58] lehetőséget nyújt az elemzések kivitelezéséhez, ez az alaprendszer meglehetősen körülményesen használható, mivel többféle ASCII-állományban kell tárolni a futtatáshoz szükséges információkat (modell, adat, kezdőértékek), és az eredményként létrejött fájlokat további szoftverekkel kell feldolgozni.

[58] Fontos megjegyezni, hogy a JAGS a BUGS-nyelvhez nagyon nagy mértékben hasonló szintaktikát használ a modellek leírásához, azonban esetenként eltér attól.

Azonban szerencsére Martyn Plummer fejlesztett egy R-csomagot is, aminek segítségével minden a modellezéshez szükséges lépés az R-környezetben végezhető (Plummer, 2015).

```
# R

library(R2WinBUGS)
library(rjags)

# modell leírása
model = function(){
  emse.db ~ dbin(emse.rata, alom.meret)
  emse.rata ~ dbeta(a, b)
}

# adatok megadása
adatok = list(
  emse.db = 540,
  alom.meret = 1187,
  a = 1, b = 1)

# kezdeti értékek megadása
inits = function(){
  list(emse.rata = rbeta(1, adatok$a, adatok$b))
}

# a vizsgálandó node-ok meghatározása
nodok = c('emse.rata')

# modell kiírása a munkakönyvtárba
setwd('C:/bugwd')
write.model(model, 'model.jags')

# a bayesi grafikus modell létrehozása
m = jags.model(
  file = "model.jags", data = adatok,
  inits = inits, n.chains = 1, n.adapt=1000)

# szimulációk futtatása
results = coda.samples(
  model = m,
  variable.names = nodok,
  n.iter = 10000,
  thin = 1)
```

Az R-szkript értelmezése Az 6-8. sorban leírjuk a modellünket R-függvény definiálásával, a `{}` között nem R, hanem JAGS-nyelven kell megírni a kódot. A 12-16. sorban az R-nyelv szabályai szerint létrehozunk egy listát a szükséges objektumokkal (`emse.db`, `alom.meret`, `a`, `b`). A 17-19. sorban a kezdeti értékekhez írunk fel egy függvényt a `model`-nél látottakhoz hasonlóan. Azonban itt a `{}` közé nem JAGS-nyelven, hanem R-nyelven hozunk létre egy listát, ami a kezdeti értékeket tartalmazza. Ebben a listában az egyes node-oknak megadhatunk konkrét értékeket, de ahogy a példában is látszik, a node priorjának eloszlásából véletlen értéket is vehetünk. A vizsgálandó node-okat az `R2WinBUGS`-nál látott módon vektorként adjuk meg (23. sor). A 26-27. sorban megadjuk a munkakönyvtárat, illetve modellünket kiírjuk egy ASCII-fájlként (`model.bug`) a `write.model()`-függvénnyel. A `write.model()`-függvény az `R2WinBUGS` könyvtár betöltésével érhető el. A 30-32. sorok a `jags.model()`-függvény paraméterezését mutatják be, amivel létrehozzuk a bayesi grafikus modellünket az előzőleg definiált modell, kezdeti értékek és forrásadatok felhasználásával. Ahogy látható a szkriptben, itt adhatjuk meg, hogy hány láncot szeretnénk futtatni. Az eddigiekhez képest újdonság, hogy az optimalizált futás érdekében a JAGS egy úgy nevezett adaptációs szakaszt is beiktat a modellünk lefordítása után, a `jags.model()`-függvény futásába, ennek hossza az `n.adapt` argumentummal adható meg. Az adaptáció során nem MCMC-szimuláció zajlik, és az itt keletkezett adatok nem fognak szerepelni az elemzésekben. A 35-39. sorban a `coda.samples()`-függvénnyel az `n.iter` argumentumban megadott számú szimulációt végzünk. Ebben a függvényben kell megadnunk a vizsgálandó node-ok neveit vektorként (23. sor), a szimulációk számát (`n.iter`) és a ritkítás mértékét (`thin`). A függvény `mcmc.list` típusú objektumot hoz létre.

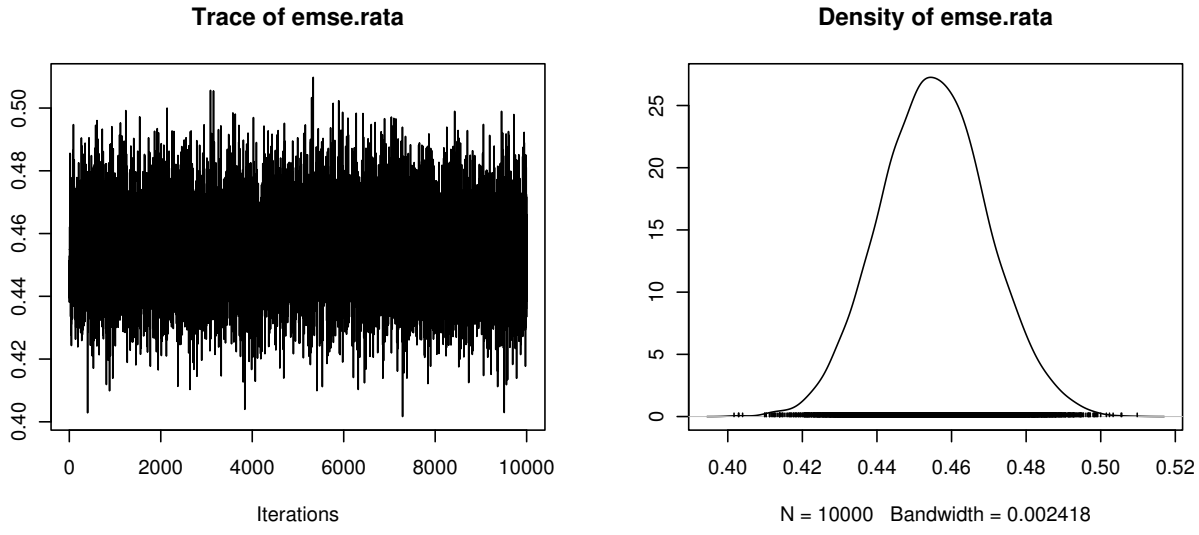
Az `mcmc.list` objektumban tárolt adatok összefoglaló táblázatának kiíratásához a `summary()` R-függvényt használhatjuk.

```
summary(results)
```

A függvény outputja az alábbi:

```
Iterations = 1:10000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 10000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean             SD       Naive SE Time-series SE
     0.4553258      0.0143921      0.0001439      0.0001439

2. Quantiles for each variable:

  2.5%    25%    50%    75%  97.5%
0.4278 0.4454 0.4552 0.4649 0.4842
```

A `plot(results)` utasítással grafikusan ábrázolhatjuk a szimulált értékeket (38. ábra: Trace of emse.rata), illetve azok eloszlását (38. ábra: Density of emse.rata).



38. ábra. A `coda.samples()`-függvénnyel létrehozott `mcmc.list` objektum ábrája, amit a `plot()`-függvénnyel hozhatunk létre.

coda

Az R `coda`-csomagja[59] számos olyan függvényt tartalmaz, ami az MC- vagy MCMC-láncok vizsgálatát, elemzését teszi lehetővé (Plummer et al., 2006). Az `rjags` csomag `coda.samples()`-függvényével létrehozott `mcmc.list` típusú objektum felhasználásával számos, a WinBUGS-ba beépített és ott nem megtalálható eljárást is használhatunk a `coda` függvényeivel.

[59] *Convergence Diagnosis and Output Analysis for MCMC*

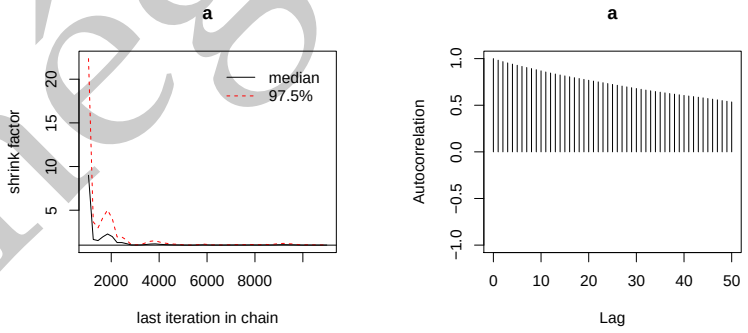
A *nem konjugált prior*-os példa (21. lap) `rjags`-ban való futtatásából származó két lánc használhatóságára vonatkozó vizsgálatok a `coda`-csomag függvényeivel az alábbi eredményeket adják. A Gelman-Rubin diagnosztika:

```
1 gelman.diag(results)
```

A függvény outputja:

```
Potential scale reduction factors:

          Point est. Upper C.I.
a               1.00       1.02
b               1.01       1.02
emse.rata       1.00       1.00

Multivariate psrf

1
```

A korábbiakban leírt módszerhez képest itt annyi különbség látható, hogy az egyes node-okra vonatkozóan nem csak az $\hat{R}$ értéket (`Point est.`) kapjuk vissza, hanem a konfidencia-intervallum felső határát (`Upper C.I.`) is. Emellett a Brooks & Gelman (1998) által bemutatott többváltozós $\hat{R}$-t kiszámolja a függvény (`Multivariate psrf`). A csomag számos ábrázolási lehetőségéből csak a korábbiakban is használt diagnosztikai eljárások közül mutatunk be két példát a 39. ábrán.



39. ábra. A `coda`-csomag `gelman.plot()`- és `autocorr.plot()`-függvényével létrehozott BGR-diagnosztikai, illetve autokorrelációs ábra.

Stan

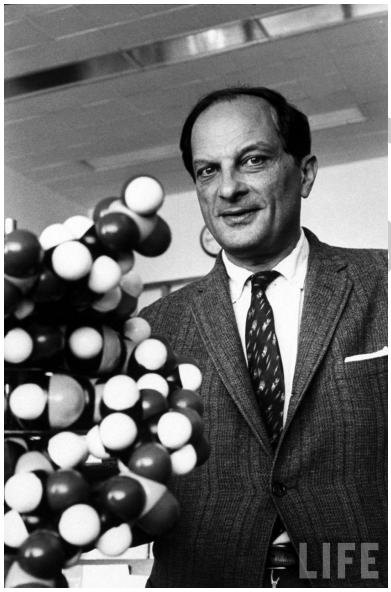
A `JAGS` futtatása során észrevehettük, hogy sokkal gyorsabban jöttek létre a láncok. Ennek oka, hogy a programot `C++` nyelven írták. Amellett, hogy a `JAGS` nagyon jó eszköz, fontos tudni, hogy *Martyn Plummer* gyakorlatilag egyedül fejleszti, dokumentálja. A függvényeket, eloszlásokat főként aszerint implementálja, hogy van-e vele tapasztalata, hasznosnak tartja-e (pl. a CAR-t nem szándékozik integrálni).

A `Stan`[60] szintén `C++` nyelven írt eszköz, ennek összes előnyével. A `JAGS`-al ellentétben azonban egy meglehetősen népes fejlesztő és dokumentáló[61] társaság gondozásában. A felhasználói támogatás egyik példája, hogy *Andrew Gelman* két, széles körben használt könyvének (Gelman & Hill, 2006; Gelman et al., 2014) minden példáját elérheti a felhasználó `Stan` szintaxisú szkriptként. Az eddig bemutatott modellezési eszközök a Gibbs-féle mintavételezési eljárást használják, amivel számos esetben nem érhető el konvergencia, ez különösen bonyolultabb hierarchikus modellek futtatása során fordulhat el. A `Stan` Hamilton-féle MC-t használ, aminek segítségével az említett esetekben is kaphatunk konvergáló láncokat. Ezeket figyelembe véve valószínűsíthető, hogy a `Stan` hosszabb távon is megbízhatóan használható lesz az elemzésekben. Így ennek az eszköznek a sajátosságait is áttekintjük röviden.

A `Stan` név *Stanisław Ulam* matematikus utónevének rövidítéséből származik. A `Stan` függvényei többek között R-hez, Pythonhoz, Matlab-hoz és Stata-hoz fejlesztett interfészeken keresztül is használhatók. A korábbi megközelítést követve itt az R-környezetben való alkalmazást mutatjuk be, amit az `rstan`-könyvtár tesz lehetővé.

A `Stan` esetén is a BUGS-nyelvhez hasonló szkript-nyelven kell leírnunk a modellünket, illetve megadni az elemzés alapjául szolgáló adatokat. Az elkészült szkriptet a modell futtatásának első lépésében a program lefordítja C++ nyelvre, amit a következő lépésben bináris állománnyá fordít le. Ehhez a folyamathoz olyan szkript-nyelvet hoztak létre a fejlesztők, amely a C++ nyelvvel rokon szintaktikailag. Az eddig használt példán mutatjuk be a Stan-szintaxist a BUGS-al kapcsolatban bemutatott ismertekkel összevetve.

A `Stan`-ben alapvető különbség, hogy nem kell foglalkoznunk a kezdőértékekkel. A `Stan`-szkriptnek minden esetben kell tartalmaznia három blokkot: `data{}`, `parameters{}` és `model{}`. Az első két blokkban a modellben használt változók típusát, illetve értéktartományát kell meghatároznunk. A harmadik blokban pedig a priorokat és likelihoodokat írjuk le. A szkript további blokkokkal bővülhet: `transformed data{}`, `transformed parameters{}`, `generated quantities{}`.



40. ábra. Stanisław Ulam (1909 – 1984)

[60] http://mc-stan.org/

[61] http://mc-stan.org/documentation/

A C++ nyelv szintaktikájának megfelelően a kód minden sorának pontosvesszővel kell végződnie. A szkriptben változók, objektumok elnevezésében nem használhatunk pontot, kötőjelet, illetve ékezetes betűt. Ha megjegyzéseket helyezünk el a szkriptben, akkor azt // jellel kell kezdenünk.

```
# R

library(rstan)

stan.kod = "
data {
  real<lower=0> a; // a béta eloszlás paramétere
  real<lower=0> b; // a béta eloszlás paramétere
  int<lower=0> emse_db;
  int<lower=0> alom_meret;
}
parameters {
  real<lower=0,upper=1> emse_rata;
}
model {
  emse_db ~ binomial(alom_meret, emse_rata);
  emse_rata ~ beta(a, b);
}
"

adatok = list(
  emse_db = 540,
  alom_meret = 1187,
  a = 1, b=1)

fit = stan(
  model_code = stan.kod,
  data = adatok,
  iter = 10000,
  chains = 2)

print(fit)

stan_trace(fit)
stan_ac(fit, pars=c('emse_rata'))
stan_hist(fit)
stan_dens(fit)
stan_plot(fit, pars=c('emse_rata'), ci_level=0.80)
```

Az R-szkript értelmezése Az rstan-könyvtár meghívásával automatikusan betöltődik a ggplot2-könyvtár is, mivel az rstan grafikai függvényei az utóbbi könyvtár függvényeit használják.

A 4-18. sorban írjuk le a korábban használt malac ivararány modellünket. Az R-ben a Stan-kódot egy szöveges objektumba írjuk, amit azzal jelzünk, hogy idézőjelek közé foglaljuk. A data{} blokkban megadjuk, hogy a modellben használt adatok milyen típusúak és milyen tartományba eshetnek. Itt a béta-eloszlás paramétereit valós számként, az emse_db, alom_meret változókat pedig egész számként adjuk meg. A parameters{} blokkban a modell által becsült paraméterek típusát és tartományát adjuk meg.

A model{} blokkban írjuk le a priorokat és a mintavételezési eloszlást. Figyelni kell, hogy az itt használt függvények neve és paraméterezése is eltér a BUGS-nyelvben látottaktól. Például a itt a dbin() helyett binomial()-t, dbeta() helyett beta()-t használunk. Azt is érdemes megjegyezni, hogy a binomial() paraméterei felcserélődtek a dbin()-hoz képest. A Stan-kódban a blokkok sorrendje nem felcserélhető, az adatoknak meg kell előznie a paramétereket, és a modellnek a kód végén kell lennie. Ha van transformed data vagy transformed parameters blokk, akkor azokat az adat illetve a paraméter blokk után kell elhelyeznünk.

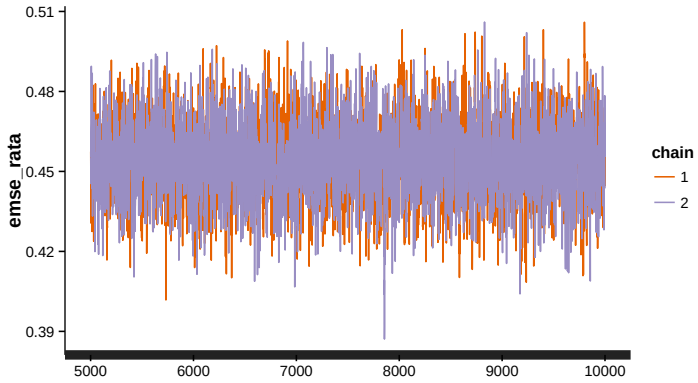
A 20-23. sorban a forrásadatok megadása a korábbiakban látottakkal azonos módon történik az R szintaxisának megfelelően, listaként.

A 25-29. sorban modell futtatását végző stan()-függvény minimális paraméterezését láthatjuk, amellyel meg kell adnunk a Stan-modell kódját, a forrásadatokat, a szimulációk és a generálandó láncok számát.

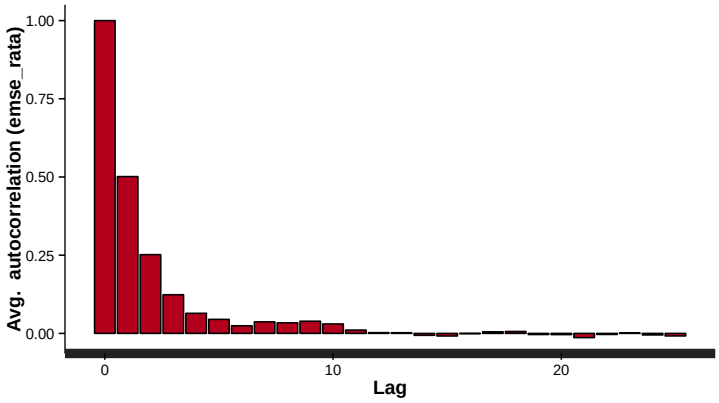
A 31. sorban látható print()-függvénnyel írathatjuk ki a modellfuttatás eredményeit. A 33-36. sorban bemutatott függvények segítségével hozhatjuk létre a generált eloszlások elemzésében használt ábrákat (41-45. ábra).

A malacok ivararányának becslése céljából Stan-ben futtatott szimulációk eredményének összefoglaló táblázata:

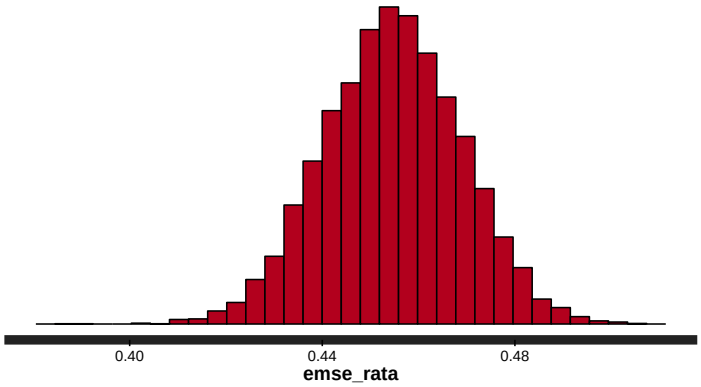
```
Inference for Stan model: 04dcac142bc0a78abd72040c6737d31e.
2 chains, each with iter=10000; warmup=5000; thin=1;
post-warmup draws per chain=5000, total post-warmup draws=10000.

             mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
emse_rata    0.46    0.00 0.01    0.43    0.45    0.45    0.46    0.48  3948    1
lp__      -819.82    0.01 0.68 -821.79 -819.99 -819.55 -819.38 -819.33  3773    1

Samples were drawn using NUTS(diag_e) at Mon Oct 19 18:58:31 2015.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```



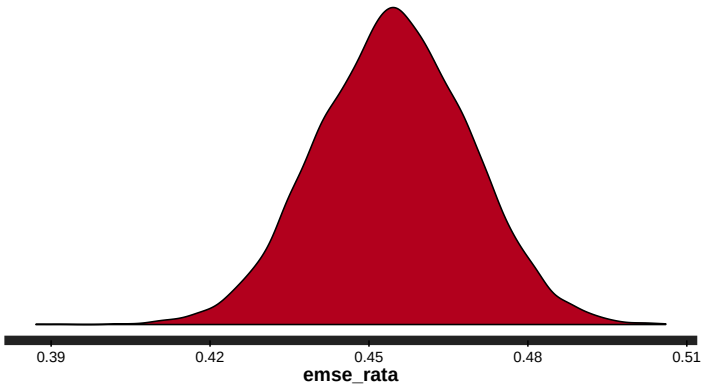
41. ábra. `stan_trace()`



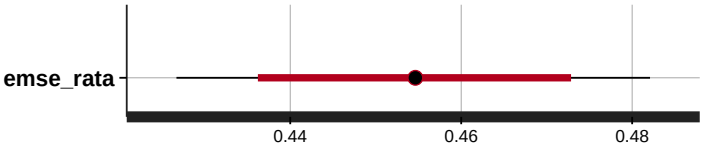
42. ábra. `stan_ac()`



43. ábra. `stan_hist()`



44. ábra. `stan_dens()`



45. ábra. `stan_plot()`

Összefoglaló táblázatok

Mintavételi eloszlás	Konjugált prior	Poszterior paraméterek	Prediktív eloszlás
$y\|\theta \sim$ Binomiális(θ, n) (Bernoulli esetén $n = 1$)	$\theta \sim$ Béta(a, n)	$a_n = a + y$ $b_n = b + n + y$	Béta-Binomiális(a_n, b_n, n)
$y\|\mu \sim \prod_{i=1}^{n}$ Normál(μ, σ^2)	$\mu \sim$ Normál $\left(\gamma, \omega^2 = \frac{\sigma^2}{n_0}\right)$	$\gamma_n = \frac{n_0\gamma + n\bar{y}}{n_0+n}$ $\omega_n^2 = \frac{\sigma^2}{n_0+n}$	Normál$(\gamma_n, \omega_n^2 + \sigma^2)$
$y\|\sigma^2 \sim \prod_{i=1}^{n}$ Normál(μ, σ^2)	$\sigma^{-2} \sim$ Gamma(a, b)	$a_n = a + \frac{n}{2}$ $b_n = b + \frac{1}{2}\sum_i (y_i - \mu)^2$	Student-t $\left(\mu, \frac{b_n}{a_n}, 2a_n\right)$
$y\|\theta \sim \prod_{i=1}^{n}$ Poisson(θ)	$\theta \sim$ Gamma(a, b)	$a_n = a + n\bar{y}$ $b_n = b + n$	NegBin $\left(\frac{b_n}{b_n+1}, a_n\right)$
$y\|\theta \sim \prod_{i=1}^{n}$ Gamma(a, b) (Exponenciális $\alpha = 1$)	$\theta \sim$ Gamma(a, b)	$a_n = a + n\alpha$ $b_n = b + n\bar{y}$	Gamma-Gamma(a_n, b_n, α)
$y\|\theta \sim \prod_{i=1}^{n}$ Uniform$(0, \theta)$	$\theta \sim$ Pareto(a, b)	$a_n = a + n$ $b_n = max\{b, y\}$	$\tilde{y} \leq b_n : \frac{a_n}{a_n+1}$Uniform$(0, b_n)$ $\tilde{y} > b_n : \frac{1}{a_n+1}$Pareto$(a_n, b_n)$
$y\|\theta \sim$ NegBin(θ, r) (Geometrikus $r = 1$)	$\theta \sim$ Béta(a, n)	$a_n = a + r$ $b_n = b + y$	NegBin-Béta(a_n, b_n, r_p)
$y\|\theta \sim \prod_{i=1}^{n}$ Pareto(θ, c)	$\theta \sim$ Gamma(a, b)	$a_n = a + n$ $b_n = b + \sum_{i=1}^{n} \log\left(\frac{y_i}{c}\right)$	$\frac{\Gamma(a_n+1)}{\Gamma(a_n)} \frac{1}{b_n\tilde{y}} \left[1 + \frac{1}{b_n} \log\left(\frac{\tilde{y}}{c}\right)\right]^{-(a_n+1)}$
$y\|\theta \sim \prod_{i=1}^{n}$ Pareto(α, θ)	$\theta \sim$ Pareto(a, b)	$a_n = a - n\alpha$ $b_n = b$ csonkolva: $(b, u = \min\{y\})$	$\frac{a_n\alpha}{a_{n+1}(b^{-a_n} - u^{-a_n})} \times$ $\tilde{y} < u : [b^{-a_{n+1}} - \tilde{y}^{-a_{n+1}}]\tilde{y}^{-(\alpha+1)}$ $\tilde{y} \geq u : b^{-a_{n+1}} - u^{-a_{n+1}}$ $a_{n+1} = a_n - \alpha$

1. táblázat. Egyváltozós konjugált prior eloszlások különböző egyparaméteres likelihoodokhoz (Lunn et al., 2012)

Eloszlás	BUGS-függvény	Sűrűség
Diszkrét egyváltozós		
Bernoulli	`r ~ dbern(p)`	$p^r(1-p)^{1-r};\ r=0,1$
Binomiális	`r ~ dbin(p, n)`	$\frac{n!}{r!(n-r)!}p^r(1-p)^{n-r};\ r=0,...,n$
Kategóriás	`r ~ dcat(p[])`	$p[r];\ r=1,2,...,dim(p);\ \sum_i p[i]=1$
Negatív binomiális	`x ~ dnegbin(p, r)`	$\frac{(x+r-1)!}{x!(r-1)}p^r(1-p)^x;\ x=0,1,2,...$
Poisson	`r ~ dpois(lambda)`	$e^{-\lambda}\frac{\lambda^r}{r!};\ r=0,1,...$
Folytonos egyváltozós		
Béta	`p ~ dbeta(a, b)`	$p^{a-1}(1-p)^{b-1}\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)};\ 0<p<1$
χ^2	`x ~ dchisqr(k)`	$\frac{2^{-k/2}x^{k/2-1}e^{-x/2}}{\Gamma(\frac{k}{2})};\ x>0$
Dupla hatvány	`x ~ ddexp(mu, tau)`	$\frac{\tau}{2}exp(-\tau\|x-\mu\|);\ -\infty<x<\infty$
Hatvány	`x ~ dexp(lambda)`	$\lambda e^{-\lambda x};x>0$
Gamma	`x ~ dgamma(r, mu)`	$\frac{\mu^r x^{r-1}e^{-\mu x}}{\Gamma(r)};\ x>0$
Általánosított gamma	`x ~ gen.gamma(r, mu, beta)`	$\frac{\beta}{\Gamma(r)}\mu^{\beta r}x^{\beta r-1}exp[-(\mu x)^\beta];\ x>0$
Log-normális	`x ~ dlnorm(mu, tau)`	$\sqrt{\frac{\tau}{2\pi}}\frac{1}{x}exp(-\frac{\pi}{2}(log(x)-\mu)^2);\ x>0$
Logisztikus	`x ~ dlogis(mu, tau)`	$\frac{\tau exp(\tau(x-\mu))}{(1+exp(\tau(x-\mu)))^2};\ -\infty<x<\infty$
Normális	`x ~ dnorm(mu, tau)`	$\sqrt{\frac{\tau}{2\pi}}exp(-\frac{\tau}{2}(x-\mu)^2);\ -\infty<x<\infty$
Pareto	`x ~ dpar(alpha, c)`	$\alpha c^\alpha x^{-(\alpha+1)};\ x>c$
Student-t	`x ~ dt(mu, tau, k)`	$\frac{\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})}\sqrt{\frac{\tau}{k\pi}}\left[1+\frac{\tau}{k}(x-\mu)^2\right]^{-(k+1)/2}$ $-\infty<x<\infty;\ k\geq 2$
Uniform	`x ~ dunif(a, b)`	$\frac{1}{b-a};\ a<x<b$
Weibull	`x ~ dweib(v, lambda)`	$\nu\lambda x^{\nu-1}exp(-\lambda x^\nu);\ x>0$
Diszkrét többváltozós		
Multinomiális	`x[] ~ dmulti(p[], N)`	$\frac{(\sum_i x_i)!}{\prod_i x_i!}\prod_i p_i^{x_i};\ \sum_i x_i=N; 0<p_i<1;\ \sum_i p_i=1$
Folytonos többváltozós		
Dirichlet	`p[] ~ ddirch(alpha[])`	$\frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)}\prod_i p_i^{\alpha_i-1};\ 0<p_i<1;\ \sum_i p_i=1$
Többváltozós normális	`x[] ~ dmnorm(mu[], T[,])`	$(2\pi)^{-d/2}\|T\|^{1/2}exp\left(-\frac{1}{2}(x-\mu)'T(x-\mu)\right)$ $-\infty<x<\infty$
Többváltozós Student-t	`x[] ~ dmt(mu[], T[,], k)`	$\frac{\Gamma((k+d)/2)}{\Gamma(k/2)k^{d/2}\pi^{d/2}}\|T\|^{1/2}\times\left[1+\frac{1}{k}(x-\mu)'T(x-\mu)\right]^{-(k+d)/2}$ $-\infty<x<\infty;\ k\geq 2$
Wishart	`x[,] ~ dwish(R[,], k)`	$\|R\|^{k/2}\|x\|^{(k-p-1)/2}exp\left(-\frac{1}{2}Tr(Rx)\right)$

2. táblázat. A WinBUGS eloszlás-függvényei (Spiegelhalter et al., 2007)

`abs(e)`	|e|
`cloglog(e)`	ln(-ln(1 - e))
`cos(e)`	cosine(e)
`cut(e)`	cuts edges in the graph - see Useofthe"cut"function
`equals(e1, e2)`	1 if e1 = e2; 0 otherwise
`exp(e)`	exp(e)
`inprod(v1, v2)`	Siv1i v2i
`interp.lin(e, v1, v2)`	v2p + (v2p + 1 - v2p) * (e - v1p) / (v1p + 1 - v1p) where the elements of v1 are in ascending order and p is such that v1p < e < v1p + 1
`inverse(v)`	v-1 for symmetric positive-definite matrix v
`log(e)`	ln(e)
`logdet(v)`	ln(det(v)) for symmetric positive-definite v
`logfact(e)`	ln(e!)
`loggam(e)`	ln(G(e))
`logit(e)`	ln(e / (1 - e))
`max(e1, e2)`	e1 if e1 > e2; e2 otherwisev
`mean(v)`	n-1Sivi n = dim(v)
`min(e1, e2)`	e1 if e1 < e2; e2 otherwise
`phi(e)`	standard normal cdf
`pow(e1, e2)`	e1e2
`sin(e)`	sine(e)
`sqrt(e)`	e1/2
`rank(v, s)`	number of components of v less than or equal to vs
`ranked(v, s)`	the sth smallest component of v
`round(e)`	nearest integer to e
`sd(v)`	standard deviation of components of v (n - 1 in denominator)
`step(e)`	1 if e >= 0; 0 otherwise
`sum(v)`	Sivi
`trunc(e)`	greatest integer less than or equal to e

3. táblázat. Logikai függvények

Irodalomjegyzék

Bocian, M., Jankowiak, H., Cebulska, A., Wisniewska, J., Fratczak, K., Wlodarski, W., & Kapelanski, W. (2012). Differences in piglets sex proportion in litter and in body weight at birth and weaning and fattening results. *Journal of Central European Agriculture*, 13(3), 475–482.

Brooks, S. P. & Gelman, A. (1998). General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical*, 7, 434–455.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2014). *Bayesian Data Analysis*. Texts in Statistical Science. Boca Raton, Florida, USA: Chapman and Hall/CRC, 3rd edition. ISBN 978-1 4398 4095 5.

Gelman, A. & Hill, J. (2006). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Analytical Methods for Social Research. Cambridge University Press.

Gelman, A. & Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences (with discussion). *Statistical Science*, 7, 457–511.

Geweke, J. (1992). Evaluating the accuracy of sampling based approaches to the calculation of posterior moments. In J. O. Bernardo, J. M. Berger, A. P. Dawid, & A. F. M. Smith (Eds.), *Bayesian statistics 4* (pp. 169–194). Oxford: Oxford University Press.

Heidelberger, P. & Welch, P. D. (1981). A spectral method for confidence interval generation and run length control in simulations. *Communication of the ACM*, 24, 233–245.

Heidelberger, P. & Welch, P. D. (1983). Simulation run length control in the presence of an initial transient. *Operations Research*, 31, 1109–1144.

Lamberson, W. R., Blair, R. M., Rohde Parfet, K. A., Day, B. N., & Johnson, R. K. (1988). Effect of sex ratio of the birth litter on subsequent reproductive performance of gilts. *J. Anim. Sci.*, 66, 595–598.

Lunn, D., Jackson, C., Best, N., Thomas, A., & Spiegelhalter, D. (2012). *The BUGS Book - A Practical Introduction to Bayesian Analysis*. Texts in Statistical Science. Boca Raton, Florida, USA: Chapman and Hall/CRC. ISBN 978-1-58488-849-9.

Plummer, M. (2015). *rjags: Bayesian Graphical Models using MCMC*. R package version 3-15.

Plummer, M., Best, N., Cowles, K., & Vines, K. (2006). CODA: Convergence Diagnosis and Output Analysis for MCMC. *R News*, 6(1), 7–11.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Raftery, A. & Lewis, S. (1992). How many iterations in the gibbs sampler? In J. O. Bernardo, J. M. Berger, A. P. Dawid, & A. F. M. Smith (Eds.), *Bayesian statistics 4* (pp. 763–773). Oxford: Oxford University Press.

Spiegelhalter, D., Thomas, A., Best, N., & Lunn, D. (2007). *WinBUGS User Manual*. 1.4.3. verzió.

Sturtz, S., Ligges, U., & Gelman, A. (2005). R2WinBUGS: A Package for Running WinBUGS from R. *Journal of Statistical Software*, 12(3), 1–16.