

افزودن توابع ریاضی و توزیع‌های آماری جدید در نرم‌افزار WinBUGS

علی‌اکبر راسخی^۱

تاریخ دریافت: ۱۳۹۷/۵/۳۰

تاریخ پذیرش: ۱۳۹۷/۸/۲۳

چکیده:

نرم‌افزار WinBUGS یکی از نرم‌افزارهای معروف در آمار بیزی محاسباتی است که می‌توان با استفاده از آن به سادگی مدل‌های بیزی را به داده‌ها برازش داد. با وجود این که توابع ریاضی و توزیع‌های معروف آماری در این نرم‌افزار به صورت تعریف شده وجود دارد، گاهی لازم می‌شود توابع و توزیع‌های دیگری را در محاسبات وارد کرد. این کار با ترفندهایی و به طور غیرمستقیم انجام می‌شود. با استفاده از ابزار توسعه WinBUGS که WBDDev نام دارد، می‌توان توابع ریاضی و توزیع‌های جدید را به این نرم‌افزار افزود و به طور مستقیم از آنها استفاده کرد. این کار نوشتن کدهای مدل را ساده‌تر و محاسبات را سریع‌تر و کاراتر می‌سازد. در این مقاله، روش و مراحل تعریف توابع و توزیع‌های جدید همراه با مثال‌ها شرح داده می‌شود.

واژه‌های کلیدی: بیزی محاسباتی، توزیع‌های آماری، درست‌نمایی، توزیع پیشینی، توزیع پسینی.

۱ مقدمه

مدل‌های بیزی را می‌توان به سادگی در آن اجرا کرد. در این نرم‌افزار توزیع‌های معروف و متداول آماری تعریف شده‌اند و با یک تابع در مدل بیزی معرفی می‌شوند. فهرست این توزیع‌ها را می‌توان در راهنمای این نرم‌افزار مشاهده کرد.^۲

اما طبیعی است که گاهی برای توزیع شرطی داده‌ها یا توزیع پیشینی پارامتر نیاز به توزیع‌های دیگری است. در این حالت‌ها اغلب از ترفندهایی استفاده می‌شود و به طور غیرمستقیم توزیع دلخواه در مدل بیزی وارد می‌شود که در بخش ۲ به این روش پرداخته خواهد شد. همچنین گاهی در نوشتن مدل بیزی به توابع ریاضی خاصی نیاز است که باید آنها را بر حسب توابع تعریف شده نوشت^۳ و برای انجام این کار ممکن است نیاز به عبارت‌های طولانی برای یک تابع باشد. بنا بر این اگر در طول برنامه نیاز به چندبار اجرای این تابع باشد، کدها طولانی‌تر خواهند شد.

اگر بتوان توزیع جدید را به طور مستقیم در مدل تعریف و استفاده کرد، نوشتن مدل ساده‌تر می‌شود. زیرا به جای نوشتن کدهایی که توزیع را به طور غیرمستقیم و از طریق ترفندها

در مسائل پارامتری استنباط آماری فرض می‌شود داده‌ها از یک توزیع آماری پارامتری آمدند و هدف برآورد این پارامتر با استفاده از داده‌هاست. در رهیافت بیزی برای پارامتر θ توزیع پیشینی $\pi(\theta)$ در نظر گرفته می‌شود و پس از مشاهده داده‌های y توزیع پسینی پارامتر به دست می‌آید و در نتیجه استنباط در مورد پارامتر انجام می‌گیرد. اگر توزیع داده‌های $y = (y_1, \dots, y_n)$ به شرط پارامتر با $f(y|\theta)$ نشان داده می‌شود، توزیع پسینی به صورت

$$\pi(\theta|y) \propto f(y|\theta)\pi(\theta) \quad (1)$$

به دست می‌آید. اما در بسیاری موارد و به ویژه مسائل کاربردی، توزیع پسینی پارامترها با روش‌های تحلیلی به دست نمی‌آید و روش‌های محاسباتی به کار گرفته می‌شوند. این محاسبات با روش‌هایی مانند مونت کارلوی زنجیرمارکوفی یا نمونه‌گیری گیبز و به کمک نرم‌افزارها انجام می‌گیرد.

یکی از این نرم‌افزارها WinBUGS است که بسیاری از

^۱ هیأت علمی گروه آمار زیستی، دانشکده علوم پزشکی، دانشگاه تربیت مدرس، ایران

^۲ فهرست این توزیع‌ها از این مسیر قابل مشاهده است: Help/User Manual/Distributions

^۳ فهرست این توابع از این مسیر قابل مشاهده است: Help/User Manual/Model Specification/Table I: Functions

برای مثال، اگر توزیع پیشینی نمایی با نرخ $1/10$ و داده‌ها به‌صورت زیر باشند:

$$(۴) \quad 0.26, 0.22, 0.17, 0.07, 0.28$$

توزیع پسینی توزیع گاما با پارامترهای $(6, 0.3282)$ خواهد شد و بنا بر این امیدریاضی θ برابر با 18.28 و انحراف معیار آن برابر با 7.46 است.

برای اجرای این مدل بیزی در نرم‌افزار، از مسیر File/New

یک فایل جدید باز کرده، کدها به‌صورت زیر وارد می‌شوند:

```
model {
  # likelihood
  for (i in 1:n) {
    y[i] ~ dweib(2,theta)
  }
  # prior
  theta ~ dexp(b)
}

# data
list(n=5, b=0.1, y=c(0.26, 0.22, 0.17, 0.07, 0.28))

# init
list(theta=10)
```

حلقه for تابع درست‌نمایی را نشان می‌دهد که تابع چگالی داده‌های $y[i]$ در آن به‌صورت وایبل با پارامتر شکل ۲ و پارامتر مقیاس θ مشخص شده است. توزیع پیشینی θ نیز نمایی با پارامتر b تعیین شده است. اولین داده‌ها را نشان می‌دهد و دومین $list$ نیز نقطه شروع تولید نمونه از توزیع پسینی است که تعیین آن اختیاری است.

برای اجرای مدل Model/Specification... را انتخاب کرده، روی $check\ model$ کلیک کنید. سپس کلمه $list$ داده‌ها را انتخاب و $load\ data$ و پس از آن $compile$ را کلیک کنید. برای تعیین نقطه شروع کلمه $list$ مربوط را انتخاب و $load\ inits$ را کلیک کنید. روش دیگر برای این که خود نرم‌افزار نقطه شروع را تولید کند $gen\ inits$ را انتخاب کنید.

در گام بعد Inference/Samples... را انتخاب کرده تا پنجره Sample Monitor Tool باز شود. مقدار $node$ را نام متغیر مورد نظر یعنی θ وارد کرده، روی set کلیک کنید^۴ و بعد برای $node$ یک * قرار می‌دهیم. سپس Model/Update... را انتخاب کرده، روی $update$ کلیک کنید. مقدار $updates$

محاسبات وارد کنند، توزیع به‌سادگی با یک تابع تعریف می‌شود. همچنین می‌توان توابع ریاضی دلخواه را فقط یک بار بر پایه توابع ریاضی تعریف‌شده نوشت و آنها را در کدهای نرم‌افزار و به‌صورت مستقیم به کار گرفت.

در ادامه به‌طور اجمالی به اجرای یک مدل بیزی و همچنین ترفندهای معرفی توزیع دلخواه پرداخته می‌شود. سپس روش افزودن توابع ریاضی و توزیع جدید مطرح می‌شود.

۲. مروری بر نرم‌افزار

برای اجرای یک مدل بیزی، باید سمت راست رابطه (۱) را در نرم‌افزار وارد کنیم تا بر اساس آن از توزیع پسینی شبیه‌سازی کند. در حالتی که y_i ها مستقل فرض شوند، تابع درست‌نمایی به‌صورت حاصل‌ضرب تابع چگالی‌ها نوشته می‌شود. پس از نوشتن مدل و معرفی مقادیر داده‌ها به نرم‌افزار، با استفاده از الگوریتم‌های مونت کارلوی زنجیرمارکوفی مشاهداتی از توزیع پسینی شبیه‌سازی خواهد شد. با داشتن این مقادیر می‌توان در مورد توزیع پسینی پارامتر کمیت‌های دلخواه را تقریب زد و به استنباط آماری پرداخت. این مراحل با یک مثال شرح داده می‌شود.

۱.۲. اجرای یک مدل بیزی

فرض کنید یک نمونه تصادفی به اندازه n از تابع چگالی زیر که رایلی نامیده می‌شود و حالت خاص توزیع وایبل به‌ازای پارامتر شکل برابر با ۲ است تولید شده است:

$$f(y|\theta) = 2\theta y e^{-\theta y^2}, \quad y > 0 \quad (۲)$$

و توزیع پیشینی پارامتر نیز توزیع نمایی با پارامتر نرخ ثابت b به‌صورت زیر در نظر گرفته شده است:

$$f(\theta) = b e^{-b\theta}, \quad \theta > 0 \quad (۳)$$

به‌سادگی می‌توان نشان داد که توزیع پسینی متناسب با $\theta^n e^{-(\sum y_i^2 + 1)\theta}$ و بنا بر این دارای توزیع گاما با پارامترهای $(n+1, \sum y_i^2 + b)$ است.

^۴ در مواردی که استنباط در مورد چند متغیر انجام می‌شود این روند به‌ازای هر کدام انجام می‌شود.

بنا بر این مدل به صورت زیر نوشته می شود.

```
model {
  # Constant
  C <- 3
  # likelihood
  for (i in 1:n) {
    z[i] <- 0
    z[i] ~ dpois(phi[i])
    phi[i] <- C - log(2*theta) - log(y[i]) +
      theta*y[i]*y[i]
  }
  # prior
  theta ~ dexp(b)
}
```

جدول ۱ نتیجه سه روش ذکر شده را نشان می دهد. در اجرای دو مدل محاسباتی تعداد تکرارها 10^4 گرفته شده است و خطای MC در این جدول از تقسیم خطای معیار بر آورد به جذر تعداد تکرارها به دست آمده است.

همان گونه که مشاهده می شود با استفاده از این ترفند، انحراف معیار و خطای مونت کارلوی بر آورد بیشتر می شود. در مدل های پیچیده تر و با تعداد زیادتر متغیرها این تفاوت بیشتر نیز خواهد بود. به همین دلیل برای مهار خطا تا یک میزان معین، لازم است برنامه با تکرارهای بیشتری اجرا شود [۴]. برای نمونه ای دیگر از مقایسه روش مستقیم و ترفند صفرها به بخش ۹.۵ مرجع [۴] مراجعه شود.

۳.۲ ترفند یک ها

در ترفند یک ها تعداد n مشاهده مستقل $z_i = 1$ از توزیع برنولی با پارامتر p_i در نظر گرفته می شود. بنا بر این تابع چگالی هر مشاهده $p_i = P(Z_i = 1)$ است و p_i برابر L_i/C قرار داده می شود که C مقادیر مثبت و به اندازه کافی بزرگ است تا اطمینان حاصل شود که مقدار L_i/C در بازه $(0, 1)$ قرار گیرد. برای جزئیات بیشتر به راهنمای نرم افزار و [۴] مراجعه شود.

تعداد نمونه تولید شده را نشان می دهد که می توان آن را تغییر داد. پس از آن در پنجره Sample Monitor Tool گزینه stats را انتخاب کنید تا آماره های میانگین، انحراف معیار و غیره در مورد متغیر مورد نظر نمایش داده شوند. البته باید توجه داشت که شرایط همگرایی در تولید نمونه ها برقرار باشد. در این مورد به مراجع [۱، ۷] مراجعه شود.

۲.۲ ترفند صفرها

اگر توزیعی در نرم افزار تعریف نشده باشد، آن گاه می توان با ترفندهایی آن را در مدل وارد کرد. تابع درست نمایی هر مشاهده y_i را با L_i نشان دهند ($i = 1, \dots, n$)، ترفند موسوم به ترفند صفرها به این صورت است که تعداد n مشاهده مستقل $z_i = 0$ از توزیع پواسون با پارامتر ϕ_i در نظر گرفته می شود. بنا بر این تابع چگالی هر مشاهده $P(Z_i = 0) = e^{-\phi_i}$ است و اگر ϕ_i برابر $-\log L_i$ قرار داده شود، تابع درست نمایی مورد نظر حاصل می شود. البته چون ϕ_i پارامتر توزیع پواسون است و باید مثبت باشد، ممکن است لازم شود عبارت $-\log L_i$ با یک مقدار ثابت C جمع شود تا همه مقادیر مثبت شوند. شایان توجه است که با این روش مقدار DIC نیز با $2nC$ جمع می شود و برای استفاده از این معیار باید DIC به دست آمده با این روش را منهای $2nC$ کرد.

برای نمونه فرض کنید در بخش پیش بدون استفاده از تابع $dweib()$ و با این ترفند مدل نوشته شود. درست نمایی مشاهده i -ام برابر است با:

$$L_i = f(y_i|\theta) = 2\theta y_i e^{-\theta y_i^2}, \quad (5)$$

که منهای لگاریتم آن محاسبه، با یک ثابت جمع و برابر با ϕ_i قرار داده می شود.

$$\phi_i = C - \log 2\theta - \log y_i + \theta y_i^2. \quad (6)$$

جدول ۱. نتایج تحلیلی و محاسباتی مثال توزیع رایلی با توزیع پیشینی نمای

روش	میانگین	خطای معیار	خطای MC
تحلیلی (دقیق)	۱۸٫۲۸	۷٫۴۶۳	۰٫۰۷۴۶۳
تابع $dweib()$	۱۸٫۲۵	۷٫۴۶۸	۰٫۰۷۲۹۶
ترفند صفرها	۱۸٫۳۲	۷٫۴۷۸	۰٫۰۸۵۶۹

۳ افزودن توابع و توزیع‌های جدید

قبل از تعریف توابع ریاضی و یا توزیع‌های جدید مراحل زیر را به ترتیب انجام دهید [۲]:^۵

۱. برنامه BlackBox را بارگزاری و نصب کنید.^۶ این برنامه بر مبنای زبان برنامه نویسی پاسکال و برای افزودن مؤلفه‌هایی^۷ به نرم‌افزار WinBUGS به کار می‌رود.

۲. فایل `wbdev_patch.zip` را بارگزاری کرده^۸، آن را از حالت فشرده خارج کنید که نتیجه فایل `wbdev_patch.txt` خواهد بود. نرم‌افزار WinBUGS را اجرا کنید و فایل متنی بالا را در آن باز کنید. حال از منوی Tools گزینه Decode را انتخاب و Decode All را اجرا کنید. با این کار برای ایجاد پوشه‌های جدید سؤال می‌شود که در پاسخ تأیید کنید. پس از این مرحله یک پوشه با نام WBDev ساخته می‌شود.

۳. کل محتویات پوشه محل نصب WinBUGS را انتخاب و در محل نصب BlackBox کپی کنید.^۹ از این پس با این پوشه کار کرده و WinBUGS را از درون آن و با کلیک روی فایل اجرایی مربوط راه‌اندازی کنید.

۱.۳ افزودن تابع ریاضی

برای تعریف تابع جدید مراحل زیر انجام می‌شود.

۱. فایل

`WBDev/Mod/ScalarTemplate.odc`

را در محیط BlackBox باز کنید. این فایل یک نمونه برای تعریف یک تابع اسکالر است^{۱۰} این فایل را با نام

^۵ مرجع [۲] از مسیر `WBDev/Docu/WBdev_functions.eps` در دسترس است.

^۶ برنامه BlackBox Component Builder از صفحه `http://www.oberon.ch/blackbox.html` به صورت یک فایل اجرایی (`.exe`) در دسترس است. با

اجرای این فایل برنامه را در مسیری مانند `C:\BUGS` نصب کنید.

^۷ `components`

^۸ این فایل را از صفحه `http://winbugs-development.mrc-bsu.cam.ac.uk/download_wbdev.html` می‌توانید دریافت کنید.

^۹ به طور پیش فرض `C:/Program files/WinBUGS14` است.

^{۱۰} برای تعریف تابع برداری می‌توان از فایل `VectorTemplate.odc` استفاده کرد.

^{۱۱} این توابع با `Math` آغاز می‌شوند (مانند `Math.Exp()`، `Math.Ln()` و غیره). فهرستی از این توابع را می‌توان با راست کلیک روی `Math` در خط `(*)2`

و انتخاب Documentation مشاهده کرد.

دلخواهی ذخیره کنید تا تغییرات را روی آن انجام دهید. برای تعریف تابع باید فقط کدهای آبی رنگ تغییر داده شوند؛ اعداد قرمز رنگ فقط شماره خط را نشان می‌دهند. توجه شود که دستورها با علامت نقطه ویرگول جدا می‌شوند و در آخرین دستور هم نقطه قرار دارد.

در دو خط ابتدا و انتها که با شماره `(*1*)` مشخص شده‌اند نام تابع را به نام دلخواه تغییر دهید. دقت شود که حرف اول نام باید بزرگ باشد. در تعریف توابع می‌توان از توابع پایه ریاضی استفاده کرد^{۱۱}.

در خط `(*5*)` عبارت زیر نوشته شده:

```
args := "vss";
```

که به این معنی است که تابع سه ورودی دارد؛ ورودی اول بردار `(v)` و ورودی‌های دوم و سوم اسکالر `(s)` هستند. در خط `(*9*)` نام این سه ورودی عبارت زیر نوشته شده است:

```
parameters = 0; dose = 1; time = 2;
```

توجه به این نکته مهم است که در این جا شمارنده از صفر شروع می‌شود یعنی ورودی اول متناظر با مقدار 0 است و به همین صورت ورودی‌های دوم و سوم با 1 و 2 تعیین می‌شوند. در خط `(*11*)` متغیرهای محلی به‌ازای ورودی‌های تابع تعریف می‌شوند. برای مثال اولین ورودی که یک بردار به نام `parameters` است، سه عضو دارد که با سه متغیر `v`، `ke`، `ka` تعریف می‌شوند. این متغیرها در خط‌های `(*13*)` تا `(*15*)` به ترتیب و به‌عنوان سه عضو بردار ورودی تابع معرفی شده‌اند. دو ورودی اسکالر دیگر نیز به همین صورت مقدار دهی شده‌اند.

در خط (*5) مقدار "sv" به این معنی است که ورودی اول اسکالر (متناظر با x) و ورودی دوم بردار (متناظر با a) است. در خط (*11) متغیرهای محلی برای اسکالر و سه عضو بردار معرفی شده‌اند و در خطوط (*13) تا (*16) این متغیرها به دو ورودی اصلی تابع منتسب شده‌اند (اندیس از 0 آغاز می‌شود). در خطوط بعد مقدار تابع محاسبه شده است که در آن تابع لگاریتم $\text{Math.Ln}()$ و دستورهایی شرط به کار رفته است.

۲. فایل `WBDev/Rsrc/Functions.odc` را باز کنید و تابع تعریف شده را با یک نام دلخواه به صورت زیر به آن اضافه کنید:

```
s <- "pql"(s) "WBDevPosQuadLn.Install".
```

توجه شود که این تابع در WinBUGS به صورت `pql()` فراخوانی خواهد شد.

۳. برای این که این تابع قابل استفاده شود، `BlackBox` را بسته و دوباره اجرا کنید.

پس از انجام این مراحل، می‌توان در برنامه‌ها از این تابع استفاده کرد. برای نمونه در کدهای زیر Z دارای توزیع یکنواخت روی بازه $(-1, 1)$ و بردار $b = (1, 5, 2)$ به ترتیب به عنوان ورودی اول و دوم این تابع در نظر گرفته شده است. بنا بر این امید ریاضی تابع به صورت زیر است:

$$E[pql(Z)] = \frac{1}{2} \int_{-1}^1 \log(1 + 5x + 2x^2) I_{(1+5x+2x^2 > 0)} dx \quad (8)$$

که با میانگین مقادیر E به روش مونته کارلو تقریب زده می‌شود. پس از اجرای مدل مقدار `mean` در پنجره `stat` این تقریب را نشان می‌دهد. مقدار دقیق تا چهار رقم اعشاری ۴/۲۲۲۸ است.

```
model {
  z ~ dunif(-1,1)
  E <- pql(z, b[])
}
```

```
list(b=c(1,5,2))
```

ادامه خطوط تا (*26) کدهای برنامه نویسی آشنا هستند که می‌توانید آنها را ویرایش کنید.^{۱۲}

پس از اتمام ویرایش `Ctrl+K` را بزنید تا برنامه `compile` شود. در این مرحله است که خطاهای احتمالی مشخص می‌شوند و می‌توان آنها را رفع کرد. اگر دستورات درست نوشته شده باشند، آن‌گاه پایین پنجره پیغام `ok` ظاهر می‌شود و در صورتی که اشکالاتی وجود داشته باشد تمام آن اشکالات با علامت ضربدر مشخص خواهد شد.

برای نمونه x را یک عدد حقیقی و $a = (a_0, a_1, a_2)$ را یک بردار از سه مقدار حقیقی در نظر می‌گیریم. می‌خواهیم تابعی تعریف کنیم که اگر عبارت درجه دوم $q(x, a) = a_0 + a_1x + a_2x^2$ مثبت بود از آن لگاریتم بگیرد و در غیر این صورت مقدار صفر را برگرداند. یعنی،

$$pql(x, a) = \begin{cases} \log[q(x, a)], & q(x, a) > 0 \\ 0, & q(x, a) \leq 0 \end{cases} \quad (9)$$

برای این کار یک تابع اسکالر به نام `PosQuadLn` به صورت شکل ۱ تعریف شده است (بخش‌های ویرایش شده با حروف مایل مشخص شده است).

```
(*1*) MODULE WBDevPosQuadLn;
.....
(*5*)   args := "sv";
.....
(*9*)   X = 0; A = 1;
(*10*)  VAR
(*11*)   x, a0, a1, a2, quad: REAL;
(*12*)  BEGIN
(*13*)   x := func.arguments[X][0].Value();
(*14*)   a0 := func.arguments[A][0].Value();
(*15*)   a1 := func.arguments[A][1].Value();
(*16*)   a2 := func.arguments[A][2].Value();
(*17*)   quad := a0 + a1 * x + a2 * x * x;
(*18*)   IF quad <= 0 THEN;
(*19*)     value := 0;
(*20*)   ELSE;
(*21*)     value := Math.Ln(quad);
(*22*)   END;
.....
(*1*)  END WBDevPosQuadLn.
```

شکل ۱. تعریف تابع `pql` رابطه (9)

^{۱۲} این کدها به زبان برنامه نویسی پاسکال نوشته می‌شوند.

در خط (*4*) پارامترها تعریف می‌شوند. مثلاً برای سه پارامتر توزیع به‌ترتیب زیر

shape = 0; scale = 1; exponent = 2;

برای پارامترهای اول، دوم و سوم قرار داده می‌شود (شمارنده از صفر آغاز می‌شود) و در خط عبارت زیر (*8*)

args := "sss";

به این معنی است که سه پارامتر اسکالر هستند. در خط عبارت زیر (*12*) عبارت

isDiscrete := FALSE;

یعنی توزیع گسسته نیست و در خط عبارت زیر (*13*)

canIntegrate := TRUE;

یعنی عبارت دقیق تابع توزیع در کدها نوشته خواهد شد که برای کار با مشاهدات سانسور شده مورد نیاز است (حالت غیر از این در توضیحات خطوط (*43*) تا (*53*) آمده است).

در خطوط (*17*) و (*18*) کران‌های متغیر تصادفی داده می‌شوند که برای مقدار بینهایت از INF استفاده می‌شود (شکل ۲).

در خطوط (*20*) تا (*29*) لگاریتم تابع چگالی توزیع با عنوان LogFullLikelihood تعریف می‌شود (شکل ۳).

اما از آن‌جا که نرم‌افزار همواره به مقدار دقیق تابع چگالی نیاز ندارد و می‌توان اعداد و مقادیر ثابت را حذف کرد، در خطوط (*30*) تا (*33*) می‌توان لگاریتم تابع چگالی را با حذف این مقادیر تحت عنوان LogPropLikelihood تعریف کرد. در این مثال همان مقدار دقیق گذاشته شد که تابع آن پیش‌تر تعریف شد. فراتر از این، گاهی فقط هسته تابع چگالی یعنی عبارت‌های شامل متغیر و نه پارامترها کافی است. بنا بر این، لگاریتم هسته تابع

توجه شود که چون ورودی دوم تابع برداری به نام b است، فراخوانی تابع با $b[]$ انجام می‌شود. برای تعریف توابع برداری به مرجع [۲] مراجعه کنید.

۲.۳ افزودن توزیع جدید

برای تعریف یک توزیع جدید، لازم است چند تابع مربوط به توزیع تعریف شود [۳]^{۱۳}. در این بخش به‌عنوان نمونه توزیع وایبل نمایی شده (EW)^{۱۴} تعریف می‌شود. تابع توزیع و چگالی وایبل با پارامتر شکل ν و مقیاس λ به‌ترتیب به‌صورت زیر هستند:

$$F(y|\nu, \lambda) = 1 - e^{-\lambda y^\nu}, \quad y > 0. \quad (9)$$

و

$$f(y|\nu, \lambda) = \lambda \nu y^{\nu-1} e^{-\lambda y^\nu}, \quad y > 0. \quad (10)$$

با استفاده از این توابع، تابع توزیع EW به‌صورت زیر تعریف می‌شود [۵، ۶].

$$G(y|\nu, \lambda, \alpha) = [F(y|\nu, \lambda)]^\alpha, \quad y > 0. \quad (11)$$

و بنا بر این تابع چگالی آن به‌صورت زیر است:

$$g(y|\nu, \lambda, \alpha) = \alpha f(y|\nu, \lambda) [F(y|\nu, \lambda)]^{\alpha-1}, \quad y > 0. \quad (12)$$

این توزیع را با $EW(\nu, \lambda, \alpha)$ نشان می‌دهند و در ادامه برای استفاده در نرم‌افزار WinBUGS تعریف می‌شود. برای تعریف این توزیع یک‌متغیره به‌صورت زیر عمل می‌شود (برای تعریف توزیع‌های چندمتغیره به مرجع [۳] مراجعه کنید).

۱. فایل WBDev/Mod/UnivariateTemplate.odc را در

محیط BlackBox باز کنید و با نام دلخواه دیگری ذخیره کنید تا تغییرات را روی آن انجام دهید. در این‌جا نیز باید فقط کدهای آبی رنگ را تغییر دهید. در دو خط ابتدا و انتها که با شماره (*1*) مشخص شده‌اند نام توزیع را تعیین کنید (حرف اول نام باید بزرگ باشد مانند (ExpWeibull).

^{۱۳} مرجع [۳] از مسیر WBDev/Docu/WBDev_distributions.eps در دسترس است.

^{۱۴} Exponentiated Weibull distribution

```
s ~ "dEweib"(s,s,s)I(s,s)
"WBDevExpWeibull.Install"
```

۳. برای این که این تابع قابل استفاده شود، BlackBox را بسته و دوباره اجرا کنید.

از این پس این توزیع سه پارامتری در WinBUGS به صورت dEweib() فراخوانی خواهد شد و نیازی به ترفندها نیست. برای این که کارکرد درست تابع تعریف شده را دید می توان مدل زیر را اجرا کرد:

```
model {
  x ~ dEweib(3,1,2)
  x2 <- x*x
  F1 <- step(1-x)
}
```

در این مدل فرض شده که متغیر تصادفی X دارای توزیع $EW(3, 1, 2)$ است. مقدار میانگین دو متغیر x و $x2$ پس از اجرای مدل به مقادیر دقیق آنها یعنی $E(X) = 1/0.772$ و $E(X^2) = 1/2.367$ بسیار نزدیک است. همچنین، میانگین متغیر $F1$ تابع توزیع در نقطه ۱ یعنی $G(1) = 0.3996$ را به خوبی تقریب می زند (تابع $step()$ در صورتی که ورودی آن نامنفی باشد مقدار ۱ و در غیر این صورت صفر را برمی گرداند. بنا بر این میانگین $F1$ مقدار $P(X \leq 1)$ را تقریب می زند).

۴ نتیجه گیری

در این مقاله به تعریف توابع ریاضی و توزیع های آماری جدید در WinBUGS به صورت مستقیم و بدون استفاده از ترفندها پرداخته شد. با یک بار تعریف و نوشتن کدهای مربوط، این توابع به نرم افزار افزوده می شوند و می توان مانند توابع موجود دیگر به سادگی در اجرای مدل های گوناگون از آنها استفاده کرد. مزیت این کار ساده تر شدن کدها و سرعت اجرای مدل خواهد بود. علاوه بر این خطای معیار برآوردها و خطای شبیه سازی نمونه کارلو نیز کاهش می یابد.

شایان ذکر است که نرم افزار OpenBUGS گونه منبع باز و به روزتری از WinBUGS است و هدف اول این مقاله

چگالی را می توان با نام LogPrior در خطوط (*34*) تا (*42*) تعریف کرد (در مثال عبارتهایی را که شامل x نیستند حذف شده اند)، به شکل ۴ مراجعه شود.

در خطوط (*43*) تا (*53*) عبارت دقیق تابع توزیع تعریف می شود. اگر قرار بر تعریف این تابع نباشد باید در خط (*13*) مقدار TRUE به FALSE تغییر یابد و خطوطی بین دو خط زیر

```
(*43*) PROCEDURE Cumulative
(*53*) END Cumulative;
```

قرار دارند مانند خط (*47*) با تغییر به عبارت زیر

```
(* HALT(126); *)
```

غیرفعال شوند (شکل ۵).

خطوط (*54*) تا (*71*) مربوط به تولید نمونه از متغیر تصادفی هستند. در این جا باید برای چهار حالت تولید نمونه تعریف شود: بدون سانسور، سانسورهای یک طرفه چپ و راست و سانسور دوطرفه. روش استفاده شده در مثال به شرح زیر است:

می دانیم اگر u یک مشاهده از توزیع یکنواخت روی بازه $(0, 1)$ و $F_X(\cdot)$ تابع توزیع متغیر تصادفی پیوسته X باشد، آن گاه مقدار $x = F_X^{-1}(u)$ یک مشاهده از متغیر تصادفی X خواهد بود. برای تولید مشاهده از متغیر تصادفی X که به بازه (x_L, x_R) محدود شده باشد می توان قرار داد $u_L = F_X(x_L)$ و $u_R = F_X(x_R)$ از توزیع یکنواخت روی بازه (u_L, u_R) یک مشاهده u^* تولید کرد و قرار داد $x = F^{-1}(u^*)$. برای سانسور یک طرفه چپ مقدار x_R برابر با کران بالای متغیر تصادفی و برای سانسور یک طرفه راست مقدار x_L برابر با کران پایین متغیر تصادفی قرار داده می شود (شکل ۶).

۲. فایل WBDev/Rsrc/Distributions.odc را باز کنید و توزیع تعریف شده را با یک نام دلخواه و در یک سطر به صورت زیر به آن اضافه کنید.

است) و Stan (که با R به خوبی هماهنگ می‌شود) و نیز بسته‌های گوناگون R (مانند بسته MCMCpack^{۱۵}) از هر دو نرم‌افزار OpenBUGS و WinBUGS مناسب‌تر و کاراتر هستند. روش‌ها و مثال‌های این مقاله روی چند رایانه با موفقیت اجرا شده است و فایل‌ها و مطالب تکمیلی از طریق مکاتبه با نویسنده در دسترس است.

سپاسگزاری

نویسنده از داوران محترم مقاله که با نظرات ارزنده خود بر کیفیت مقاله افزودند تشکر و سپاسگزاری می‌نماید.

کار با OpenBUGS بود. اما نویسنده برای یکپارچه کردن OpenBUGS و BlackBox به روشی رضایت بخش و مطمئن دست نیافت که بتوان روی رایانه‌هایی با نسخه‌های متفاوت سیستم عامل آن را اجرا کرد. بنا بر این از WinBUGS استفاده شد؛ زیرا اولاً هر دو نرم‌افزار بر پایه زبان پاسکال نوشته شده‌اند و روش و مراحل افزودن توابع به آنها بسیار شبیه است و ثانیاً WinBUGS به سادگی و با کپی کردن با BlackBox تلفیق می‌شود. به هر حال انجام این کار با استفاده از OpenBUGS می‌تواند موضوع پژوهش‌های بعدی باشد. هر چند برای افزودن توابع جدید در محاسبات بیزی اگر از جنبه به‌روز بودن به مسئله پرداخته شود، نرم‌افزارهایی همچون JAGS (که بر پایه C++

```
(*1*) MODULE WBDevExpWeibull;
.....
(*4*) shape = 0; scale = 1; exponent = 2;
.....
(*8*) args := "sss";
(*9*) END DeclareArgTypes;
.....
(*12*) isDiscrete := FALSE;
(*13*) canIntegrate := TRUE;
(*14*) END DeclareProperties;
.....
(*17*) lower := 0;
(*18*) upper := INF;
```

شکل ۲. تعریف نام پارامترها و کران متغیر

```
(*20*) PROCEDURE LogFullLikelihood (node: WBDevUnivariate.Node; OUT value: REAL);
(*21*) VAR
(*22*) x, nu, lambda, alpha, e1, e2, e3, e4: REAL;
(*23*) BEGIN
(*24*) x := node.value;
(*25*) nu := node.arguments[shape][0].Value();
(*26*) lambda := node.arguments[scale][0].Value();
alpha := node.arguments[exponent][0].Value();
e1 := Math.Ln(alpha) + Math.Ln(lambda) + Math.Ln(nu);
e2 := (nu-1)*Math.Ln(x);
e3 := lambda*Math.Power(x,nu);
e4 := (alpha-1)*Math.Ln(1-Math.Exp(-lambda*Math.Power(x,nu)));
(*27*) value := e1 + e2 - e3 + e4;
(*28*)
(*29*) END LogFullLikelihood;
```

شکل ۳. لگاریتم تابع چگالی دقیق توزیع

^{۱۵} برای نمونه، در این بسته تابع MCMClogit() برای رگرسیون لجستیک بیزی وجود دارد که در آن می‌توان به سادگی تابع پیشینی دلخواه را برای پارامتر تعریف کرد.

```

(*30*) PROCEDURE LogPropLikelihood (node: WBDevUnivariate.Node; OUT value: REAL);
(*31*) BEGIN
(*32*)   LogFullLikelihood(node, value);
(*33*) END LogPropLikelihood;

(*34*) PROCEDURE LogPrior (node: WBDevUnivariate.Node; OUT value: REAL);
(*35*) VAR
(*36*)   x, nu, lambda, alpha, e2, e3, e4: REAL;
(*37*) BEGIN
(*38*)   x := node.value;
   nu := node.arguments[shape][0].Value();
   lambda := node.arguments[scale][0].Value();
   alpha := node.arguments[exponent][0].Value();
   e2 := (nu-1)*Math.Ln(x);
   e3 := lambda*Math.Power(x,nu);
(*39*)   e4 := (alpha-1)*Math.Ln(1-Math.Exp(-lambda*Math.Power(x,nu)));
(*40*)   nu := node.arguments[shape][0].Value();
(*41*)   value := e2 - e3 + e4;
(*42*) END LogPrior;

```

شکل ۴. هسته تابع درست‌نمایی توزیع

```

(*43*) PROCEDURE Cumulative (node: WBDevUnivariate.Node; x: REAL; OUT value: REAL);
(*44*) VAR
(*45*)   nu, lambda, alpha: REAL;
(*46*) BEGIN
(*47*)   (* HALT(126); *)
(*48*)   nu := node.arguments[shape][0].Value();
(*49*)   lambda := node.arguments[scale][0].Value();
   alpha := node.arguments[exponent][0].Value();
(*51*)   value := Math.Power(1-Math.Exp(-lambda*Math.Power(x,nu)), alpha);
(*52*)
(*53*) END Cumulative;

```

شکل ۵. تعریف تابع توزیع

```

(*54*) PROCEDURE DrawSample (node: WBDevUnivariate.Node; censoring: INTEGER; OUT sample: REAL);
(*55*) VAR
(*56*)   nu, lambda, alpha, u, left, right, FL, FR, uL, uR, uLR: REAL;
(*57*) BEGIN
(*58*)   nu := node.arguments[shape][0].Value();
(*59*)   lambda := node.arguments[scale][0].Value();
   alpha := node.arguments[exponent][0].Value();
(*60*)   node.Bounds(left, right);
   u := WBDevRandnum.Uniform(0, 1);
   FL := Math.Power(1-Math.Exp(-lambda*Math.Power(left,nu)), alpha);
   FR := Math.Power(1-Math.Exp(-lambda*Math.Power(right,nu)), alpha);
   uL := WBDevRandnum.Uniform(FL, 1);
   uR := WBDevRandnum.Uniform(0, FR);
   uLR := WBDevRandnum.Uniform(FL, FR);
(*61*)   CASE censoring OF
(*62*)     |WBDevUnivariate.noCensoring:
(*63*)       sample := Math.Power(-1/lambda*Math.Ln(1-Math.Power(u,1/alpha)), 1/nu);
(*64*)     |WBDevUnivariate.leftCensored:
(*65*)       sample := Math.Power(-1/lambda*Math.Ln(1-Math.Power(uL,1/alpha)), 1/nu);
(*66*)     |WBDevUnivariate.rightCensored:
(*67*)       sample := Math.Power(-1/lambda*Math.Ln(1-Math.Power(uR,1/alpha)), 1/nu);
(*68*)     |WBDevUnivariate.intervalCensored:
(*69*)       sample := Math.Power(-1/lambda*Math.Ln(1-Math.Power(uLR,1/alpha)), 1/nu);
(*70*)   END;
(*71*) END DrawSample;

```

شکل ۶. تعریف روش تولید نمونه از توزیع

مراجع

- [1] Christensen, R., Johnson, W., Branscum, A. and Hanson, T. E. (2011). *Bayesian Ideas and Data Analysis: An Introduction for Scientists and Statisticians*, CRC Press, Boca Raton.
- [2] Lunn, D. (2004). WinBUGS Development Interface (WBDev)–Implementing your own functions. Imperial College School of Medicine, from WBDev/Docu/WBDev.
- [3] Lunn, D. and Jackson, C. (2004). WinBUGS Development Interface (WBDev)–Implementing your own univariate distributions. Imperial College School of Medicine, from WBDev/Docu/WBDev.
- [4] Lunn, D., Jackson, C., Best, N., Thomas, A. and Spiegelhalter, D. (2012). *The BUGS book: A practical introduction to Bayesian analysis*, CRC press, London.
- [5] Mudholkar, G. S. and Srivastava, D. K. (1993). Exponentiated Weibull family for analyzing bathtub failure-rate data, *IEEE Transactions on Reliability*, **42**(2), 299-302.
- [6] Nadarajah, S., Cordeiro, G. M. and Ortega, E. M. (2013). The exponentiated Weibull distribution: a survey, *Statistical Papers*, **54**(3), 839-877.
- [7] Ntzoufras, I. (2009). *Bayesian Modeling Using WinBUGS*, John Wiley And Sons, New Jersey.