به نام آن که جان را فکرت آموخت

دانشگاه صنعت آب و برق (شهید عباسپور)

آشنایی با زبان محاسبات آماری

(ویراست نوشتار ۱٫۸٫۰)

سیدسعید موسوی‌ندوشنی

s_mousavi@pwut.ac.ir

پاییز ۱۳۹۱

فهرست مطالب

پیش‌گفتار

کوچک اما توانا، این توصیف شاید موجزترین تعبیری باشد که می‌توان راجع به نرم‌افزار R بیان نمود. نرم‌افزار R محیط بسیار مناسبی برای محاسبات آماری و ترسیم نمودارها است. این نرم‌افزار در سال‌های اخیر در دنیا شهرت بسزایی یافته و نظر کاربران زیادی را به خود جلب نموده است. شاید بتوان عوامل زیر را به‌عنوان علت شهرت آن احصاء نمود.

- این نرم‌افزار رایگان است و غالباً افراد دانشگاهی حامیان آن هستند.
- روی اکثر سکوها[1] قابل نصب است و کار می‌کند.
- تعداد زیادی بسته نرم‌افزار[2] (متجاوز از 2000 مورد) روی آن نصب می‌شود که زمینه‌های مختلف آماری را در بر می‌گیرد و از این حیث قدرت شگرفی را به این نرم‌افزار به ظاهر کوچک می‌بخشد.

در ایران تاکنون به منظور معرفی نرم‌افزار R کارگاه‌های آموزشی مختلفی برگزار شده است. بلاگ‌های گوناگون تهیه گردیده است و چند کتاب و احیاناً جزوه و یا جزوه‌هایی به منظور آموزش آن، به رشته تحریر در آمده است. که هر یک در جایگاه خود درخور توجه و تشکر است.

نوشتار[3] حاضر کوشش کوچکی به منظور معرفی نرم‌افزار R برای خوانندگان فارسی زبان است. به رسم مستندات رایگان R و بنا بر عرف سایر کشورها، این نوشتار برای استفاده همگانی روی سایت بین‌المللی R قرار گرفته است و قابل پیاده‌سازی می‌باشد. سعی نگارنده این سطور بر آن است که در حد بضاعت اندک خود به مرور زمان،

1. platforms 2. package

۳. خیلی از مؤلفین در جهان، مستندات خود را راجع به R، با نرم‌افزار LaTeX تهیه نموده‌اند. نوشتار حاضر نیز با نرم‌افزار LaTeX-ماپرسی حروف‌نگاری شده است.

آن را کامل‌تر نماید. تا که قبول افتد و که در نظر آید.
ضمناً نوشتار دیگری زیر عنوان «مباحث ویژه در R» روی سایت R به نشانی زیر قرارگرفته است که به‌صورت رایگان قابل پیاده‌سازی[4] است.

http://cran.r-project.org/doc/contrib/Mousavi-R_topics_in_Farsi.pdf

مسلماً وجیزه حاضر خالی از خلل نیست، نویسنده از هرگونه اظهار نظر و پیشنهادی برای اصلاح و ارتقاء آن استقبال و استفاده خواهد نمود.

سیدسعید موسوی‌ندوشنی
تهران - پاییز ۱۳۹۱

4. download

فصل اول

آشنایی با زبان برنامه‌نویسی R

۱-۱ مقدمه

نرم‌افزار R یک زبان برنامه‌نویسی ریاضی شی‌گرای می‌باشد که بسیار شبیه S-plus (بسته نرم‌افزاری مشهور آماری) و برای محاسبات آماری طراحی شده است. پروژه R از سال ۱۹۹۵ در گروه آمار دانشگاه Auckland توسط آقایان Robert Gentleman و Ross Ihaka (به همین علت نام R برای آن انتخاب گردید.) شروع شد و بزودی مخاطبین زیادی یافت. در حال حاضر این زبان را یک تیم بین‌المللی نگه‌داری می‌کند و داوطلبانه توسعه می‌یابد. نشانی صفحه web پروژه R به شرح زیر است:

http://www.r-project.org

۱-۲ چرا زبان برنامه‌نویسی

قبل از این که به جزییات بیشتری پرداخته شود. این سوال پیش می‌آید که چرا نیاز به زبان برنامه‌نویسی در محیط آماری وجود دارد، ضمن این که نرم‌افزارهای مختلف آماری (spss, sas, Minitab, statistica, ...) نیز

در دسترس است. برای پاسخ به این سوال می‌توان موارد زیر را عنوان نمود.

- برای این که اساس یک روش، مدل و الگورتیم آماری را به درستی درک کنید و فقط با یک جعبه سیاه کار نکنید.
- اگر بخواهید روی خروجی خود کنترل بیشتری داشته باشید.
- اگر بخواهید برای روش‌های جدید آماری و یا الگوریتم خودتان برنامه و یا نرم‌افزاری (package, extension) را بسط دهید و فقط به روش‌های کلاسیک آماری بسنده نکنید.

محاسن این زبان به شرح زیر است.

- زبان R رایگان است و دارای متن باز می‌باشد و روی سیستم عامل‌های ویندوز[1]، یونیکس[2]، لینوکس[3] و مکینتاش[4] اجرا می‌شود.
- زبان R دارای راهنمای داخلی خوبی است.
- زبان R دارای قابلیت‌های قابل ملاحظه گرافیگی است.
- آشنایی با این زبان به منزله آشنایی با زبانِ تجاریِ آماریِ S-plus است.
- زبان R، زبانی قوی است. یادگیری آن ساده است و دارای توابع پیش‌ساخته آماری فراوانی است و packageهای بسیار زیادی به آن اضافه می‌شود.
- در این زبان به سادگی می‌توان توابع مورد نظر کاربر را ساخت.
- با زبان R می‌توان package ساخت و در محیط R از آن استفاده نمود.
- فراوانی ارتقاء این زبان قابل ملاحظه است، به عنوان مثال به شرح زیر توجه کنید.

- version 2.6.2 (2008-02-08)
- version 2.7.0 (2008-04-22)
- version 2.7.1 (2008-06-23)
- version 2.7.2 (2008-08-25)
- version 2.8.0 (2008-10-20)
- version 2.8.1 (2008-12-22)
- version 2.9.0 (2009-04-17)
- version 2.9.1 (2009-06-26)
- version 2.9.2 (2009-08-24)

1. Windows 2. Unix 3. Linux 4. Macintosh

- version 2.10.0 (2009-10-26)
- version 2.10.1 (2009-12-14)
- version 2.11.0 (2010-04-22)
- version 2.11.1 (2010-05-31)
- version 2.12.0 (2010-10-15)

نسخه 1.0.0 زبان R در سال 2000 میلادی و نسخه 2.0.0 آن در سال 2004 میلادی ارائه شده است و همان‌طور که گفته شد مرتباً به‌روز می‌گردد.

محدودیت‌های زبان R به شرح زیر است:

- دارای امکان ایجاد Gui نیست (در این مورد S-plus امکانات خوبی دارد)
- یک سیستم تجاری آن را پشتیبانی نمی‌کند.
- برای استفاده و برنامه‌نویسی با آن، باید فرامین آن را آموخت.
- زبان R دارای مفسر است، بدین معنی که برای اجرا، ترجمه نمی‌شود و به صورت فایل اجرایی (*.exe) در نمی‌آید.

۱-۳ طرز نصب نرم‌افزار

ابتدا فایل اجرایی R-2.7.0-win32.exe را از سایت R دانلود[5] نموده و با دوبارکلیک متوالی روی آن نصب می‌شود. پس از پایان عمل نصب روی desktop کامپیوتر شما یک آیکون به شکل حرف R قرار می‌گیرد. اگر روی حرف R دوبار کلیک کنید، صفحه‌ای باز می‌شود که R Console نام دارد. در این صفحه یک مقدار توضیحاتی وجود دارد و پس از آن علامت «>[6]» ملاحظه می‌شود که در مقابل آن می‌توان عملیات مورد نظر را انجام داد. البته می‌توان این علامت را تغییر داد. نحوهٔ تغییر علامت به شرح زیر است.

```
> options(prompt="R> ")
R>
```

۱-۴ شی‌ها

وقتی R اجرا می‌گردد، متغیرها، توابع، نتایج و ... در حافظه فعال کامپیوتر به شکل اشیاء با یک نام ذخیره می‌شوند. کاربر روی اشیاء و با استفاده از عملگرها (حسابی، نسبی و منطقی) و توابع (که خودشان جزء اشیاء

۵. لازم به یادآوری است که در حین این عمل ممکن است نسخه‌ی جدیدتری در دسترس شما قرار گیرد. نشانی سایت عبارتست از: http://CRAN.R-project.org

6. prompt

هستند) عمل می‌کند.
اولین کاراکتر نام یک شی حتماً باید با حروف A-Z و یا a-z شروع گردد و بقیه کاراکترها آن می‌تواند حروف، ارقام 9-0، نقطه «.» و زیرخط «_» باشد. زبان R بین حروف بزرگ و کوچک تفکیک قائل می‌شود. بنابراین بین x و X تفاوت وجود دارد.

۱-۴-۱ ایجاد، فهرست و حذف کردن اشیاء در حافظه

یک شی می‌تواند با عملگر نسبت دادن ایجاد شود. برای این کار می‌توان از علامت‌های «منها و کوچکتر <-» از استفاده نمود. به مثال زیر توجه کنید.

```
> n <- 15
```

اگر بخواهید مقدار n را مشاهده کنید کافی است که n را تایپ نموده و کلید Enter را بزنید. کنار محتوی n رقم 1 داخل کروشه ظاهر می‌گردد که اولین عنصر شی n را نشان می‌دهد. این دستور در واقع یک فرمان چاپ ضمنی ()print است. البته در مواردی باید این تابع را صریحاً فراخواند. به‌عنوان مثال می‌توان از چاپ اشیاء در حلقه و یا یک تابع نام برد.

```
> n
[1] 15
```

البته برای نسبت دادن می‌توان مانند سایر زبان‌های برنامه‌نویسی از عملگر «=» نیز استفاده نمود. اما باید توجه داشت که علامت «<-» کاربر وسیع‌تری دارد[۷]، که به یک نمونه آن، در بحث ماتریس‌ها اشاره خواهد شد.

```
> n <- 15
> n
[1] 15
```

همان‌طور که گفته شد، زبان R بین حروف کوچک و بزرگ تفاوت می‌گذارد.

```
> x <- 1
> X <- 10
> x
[1] 1
> X
[1] 10
```

می‌توان یک عبارت را به یک شی نسبت داد.

```
> n <- 10+2
> n
[1] 12
```

توجه کنید که عملیات ریاضی را می‌توان بدون استفاده از شی نیز انجام داد.

```
> (10+2)*5
[1] 60
```

۷. بدین معنی که می‌تواند به عنوان آرگومان یک تابع مورد استفاده قرارگیرد، در صورتی که علامت تساوی این خاصیت را ندارد.

اگر دستور مورد نظر در بیشتر از یک سطر تایپ شود، آنگاه در سطر و یا سطرهای بعدی، زبان R ادامه کار را با علامت «+» نشان می‌دهد. فرض کنید که مثال اخیر در دو سطر تایپ شود، نتیجه به‌صورت زیر است.

```
> (10+2)*
+ 5
[1] 60
```

مثال دیگری را ملاحظه کنید.

```
> (10+2)*
+ 5/
+ 10 -
+ 1
[1] 5
```

در دو مثال اخیر، رفتن به سطرهای بعدی (به علت کوتاهی دستورات) لزومی ندارد.
اگر شی مقدار نداشته باشد خطا حاصل می‌شود، چون اصلاً شی بدون مقدار ایجاد نمی‌گردد. به مثال زیر توجه کنید.

```
> x <- 3
> x+y
Error: object 'y' not found
```

۱-۴-۲ ویرایشگرهای زبان R

اگر کُدهای یک برنامه طولانی بود، اجرای تک‌تک دستورات خسته‌کننده است و مناسب است که از یک ویرایشگر استفاده شود. به عبارت دیگر، در این حالت قرار است که دستورات به‌صورت گروهی[8] اجرا گردند. در این ویرایشگرها فایل‌هایی از مجموعه دستورات با پسوند r ایجاد می‌شوند که قابل ذخیره شدن و بازیابی مکرر است. ویرایشگرها به دو دسته تقسیم می‌شوند.

- یک ویرایشگر داخل خود RGui است. برای این کار پس از مرحله نصب نرم‌افزار، روی آیکون R کلیک نموده تا نرم‌افزار اجرا شود و سپس در منوی بالای صفحه از منوی File گزینه New script را زده و صفحه جدیدی باز می‌شود که بالای آن R Untitled - R Editor نوشته شده است. اکنون می‌توان کُدهای مورد نظر خود را هر چندتا که هستند نوشت و سپس ذخیره نمود و بعد اجرا کرد. برای اجرای کُدها کافی است که در R Console یعنی جایی که علامت «>» وجود دارد، دستور زیر را تایپ نموده و کلید Enter را بزنید.

```
> source("file name")
```

اگر فایل در پوشه جاری نبود، آنگاه علاوه بر نام فایل باید آدرس مسیر را نیز اضافه نمود. به عنوان مثال

```
> source("E:/users/test.r")
```

8. Batch

به تغییر علامت \ به / برای بیان مسیر توجه کنید. ذکر این نکته ضروری است که از روی خود script نیز برنامه قابل اجرا است. کافی است که تمام و بخشی از کُدها را انتخاب نموده و کلید Ctrl را نگه داشته و کلید R را بزنید تا برنامه اجرا گردد.
پس از حصول اطمینان از درستی کُدهای خود می‌توانید آنها را ذخیره کنید. برای این کار کافی است که کلید Ctrl را نگه داشته و کلید S را بزنید، آنگاه پنجره‌ای به نام Save script as ظاهر می‌شود که می‌توان مسیر مورد نظر برای ذخیره را انتخاب نمود، سپس نام script خود را مشخص نموده و کلید Save را کلیک نموده به این ترتیب کُدهای شما در یک فایل با پسوند R و نام انتخابی در مسیر مورد نظر ذخیره می‌گردد. از این به بعد هر گاه به آن نیاز داشته باشید با انتخاب منوی فایل از نوار ابزار بالای صفحه R گزینه Open script را بزنید و آن را بازکنید.

- ویرایشگرهای خارجی: منظور از این ویرایشگرها آنهایی هستند که داخل نرم‌افزار نیستند و باید مستقلاً download شوند. بعضی از آنها رایگان و پاره‌ای از آنها تجاری هستند. از بین این دسته از ویرایشگرها می‌توان به Tinn-R، RWinEdt و XEmacs اشاره نمود.

ویرایشگر Tinn-R[۹]: رایگان است و پس از نصب آن می‌توان نرم‌افزار R را داخل آن صدا زد، آنگاه خودبخود صفحه کامپیوتر بدو قسمت مساوی تقسیم می‌شود و می‌توان کُدهای نوشته شده را اجرا نمود.

ویرایشگر RWinEdt: برای استفاده از ویرایشگر RWinEdt باید مراحل زیر را طی نمود.

- ابتدا باید نرم‌افزار WinEdt را نصب کرد، البته خاطر نشان می‌شود که این نرم‌افزار تجاری است.
- در این مرحله باید package و یا بسته‌ای به نام RWinEdt را download نموده و سپس آن را نصب کنید. بسته فوق جزء بسته‌های نرم‌افزاری R می‌باشد و لذا رایگان است.
- سپس در نرم‌افزار R، دستور زیر را برای یکبار اجرا کنید.

```
> library ("RWinEdt")
```

پس از اجرا، یک آیکون تحت عنوان RWinEdt روی desktop کامپیوتر ایجاد می‌شود.

- کُدهای مورد نظر را می‌توان در ویرایشگر فوق نوشت و اجرا نمود. خاطر نشان می‌سازد که قبلاً باید RGui باز شده باشد.

ویرایشگر RStudio: پس نصب نرم‌افزار R می‌توان این ویرایشگر را نصب نمود. این ویرایشگر به‌طور خودکار به R متصل می‌گردد. محیط این ویرایشگر شبیه نرم‌افزار Matlab است. برای به‌دست آوردن این ویرایشگر می‌توان به آدرس زیر مراجعه نمود.

http://www.rstudio.org/download/desktop

۱-۴-۳ داده‌ها در R

همانطورکه ملاحظه شد R با شی‌ها کار می‌کند که خود آن‌ها توسط نام و محتوی مشخص می‌شوند. هم‌چنین نوع داده که در شی قرار دارد با خصوصیت[10] معین می‌گردد. تمام شی‌ها دارای دو خصوصیت است:

- mode: نوع عناصر یک شی را مشخص می‌کند. چهار نوع اصلی mode وجود دارد: عددی، کاراکتر، مختلط و منطقی. البته modeهای دیگری نیز وجود دارد که در مورد data به‌کار نمی‌رود. برای مثال می‌توان از تابع یا عبارت نام برد.
- طول (length): تعداد عناصر یک شی را نشان می‌دهد.

اکنون به مثال‌های زیر توجه کنید.

```
> x <- 1
> mode(x)
[1] "numeric"
> length(x)
[1] 1
> A <- "Auchland"; compar <- TRUE; z <- 1i
> mode(A); mode(compar); mode(z)
[1] "character"
[1] "logical"
[1] "complex"
```

توجه داشته باشید که واژه TRUE حتماً باید با حروف بزرگ نوشته شود وگرنه با پیغام زیر روبرو می‌شوید.

توجه: همان‌طورکه قبلاً ملاحظه شد در هر سطر یک دستور می‌آید. اما اگر بخواهید که در یک سطر بیش از یک دستور قرارگیرد، آنگاه باید دستورات یک سطر، با علامت «;» از یکدیگر جدا شوند، در غیر این صورت با خطا مواجه خواهید شد.

```
Error: object "true" not found
```

خصوصیت mode در مورد داده‌های مفقود شده به صورت NA[11] نشان داده می‌شود.

زبان R مقادیر عددی نامعین، مثل $\pm\infty$ را به نوبت با Inf و Inf- نشان می‌دهد. اگر مقدار عدد نباشد، آن را با NaN[12] نمایش می‌دهد. به مثال‌های زیر توجه کنید.

```
> x <- 5/0
> x
[1] Inf
> exp(x)
[1] Inf
> exp(-x)
[1] 0
> x - x
[1] NaN
```

10. attribute 11. Not Available 12. not a number

برای ایجاد رشته کافی است که آن را داخل یک و یا دو quotes قرار دهید. به مثال‌های زیر توجه کنید.

```
> x <- "This is a test."
> x
[1] "This is a test."
> x <- 'This is a test.'
> x
[1] "This is a test."
```

توجه کنید که چه با یک و یا دو quotes عمل کنید، حاصل به صورت دو quotes است. اکنون اگر در داخل رشته آپاستورف وجود داشت، در صورت استفاده از دو quotes مشکلی وجود ندارد، اما اگر از یک quotes استفاده شود، خطا پیش می‌آید و باید از «\» استفاده نمایید.

```
> x <- "Ali's apple"
> x
[1] "Ali's apple"
> x <- 'Ali\'s apple'
> x
[1] "Ali's apple"
```

با دستور digits می‌توان تعداد ارقام اعشاری را کنترل نمود. به مثال زیر توجه کنید.

```
> print(1/1:5, digits=2)
[1] 1.00 0.50 0.33 0.25 0.20
```

البته می‌توان تعداد ارقارم اعشار را به‌طور کلی معین نمود که دیگر نیازی برای تنظیم آنها در هر دستور نباشد. برای این کار از تابع options استفاده می‌شود.

```
> options(digits = 3)
> print(1/1:5)
[1] 1.000 0.500 0.333 0.250 0.200
```

ضمناً توجه داشته باشید که تابع print() فقط یک شی را چاپ می‌کند و چنانچه مایل باشید که بیش از یک شی چاپ گردد باید از تابع cat() استفاده نمود. به مثال زیر توجه کنید.

```
> x <- 1:10
> cat('x=', x, '\n')
x= 1 2 3 4 5 6 7 8 9 10
```

قسمت '\n' برای این است که جای «>» به هم نخورد. به عبارت دقیق‌تر این فرمت خط و یا سطر تعویض می‌کند اگر بخواهید که قسمت «x=» و اعداد در دو سطر جدا چاپ شوند باید به صورت زیر عمل کنید.

```
> x <- 1:10
> cat('x=', '\n', x, '\n')
x=
1 2 3 4 5 6 7 8 9 10
```

۱-۴-۴ داده‌های آماده در زبان R

در زبان R تعدادی داده آماده برای استفاده در مثال‌ها قرار داده شده است که در این نوشتار نیز از آن استفاده می‌شود. مانند cars، iris، LakeHuron، Nile، trees و . . . البته می‌توان فهرست کامل آن‌ها را با استفاده از دستور زیر در R یافت.

```
> data()
```

ضمناً اگر از یک package استفاده شد، با دستور زیر می‌توان داده‌های همراه آن را فراخواند.

```
> library(gstat)
```

اکنون می‌توان فهرست داده‌های مندرج در بسته gstat را توسط دستور زیر ملاحظه نمود.

```
> data(package="gstat")
```

به عنوان مثال می‌توان از میان اسامی داده‌های فهرست شده، داده‌های مربوط به meuse.alt را با دستورات زیر ملاحظه نمود.

```
> data(meuse.alt)
> meuse.alt
```

۱-۴-۴-۱ نمایش قسمتی از داده‌ها درزبان R

در پاره‌ای از اوقات به سبب طولانی بودن داده‌ها، می‌توان قسمتی از آنها را مشاهده نمود. برای این کار می‌توان از دو تابع head() و tail() استفاده نمود.

- برای نشان دادن سطرهای اولیه (6 سطر) داده از تابع head() استفاده می‌شود. برای مثال داده‌های trees را در R در نظر گیرد.

```
> data(trees)
> head(trees)
```

	Girth	Height	Volume
1	8.3	70	10.3
2	8.6	65	10.3
3	8.8	63	10.2
4	10.5	72	16.4
5	10.7	81	18.8
6	10.8	83	19.7

اگر به بیش از 6 سطر و یا کمتر از 6 سطر لازم باشد، باید تعداد مورد نظر صریحاً قید گردد.

```
> head(trees, 8)
```

```
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
4  10.5     72   16.4
5  10.7     81   18.8
6  10.8     83   19.7
7  11.0     66   15.6
8  11.0     75   18.2
```

- برای نشان دادن سطرهای اواخر (6 سطر) داده از تابع ()tail استفاده می‌شود. برای مثال داده‌های trees را در R در نظر گیرد.

```
> tail(trees)
```

```
   Girth Height Volume
26  17.3     81   55.4
27  17.5     82   55.7
28  17.9     80   58.3
29  18.0     80   51.5
30  18.0     80   51.0
31  20.6     87   77.0
```

از حیث تعداد سطرهای بیشتر و کمتر از 6 سطر، تابع ()tail مانند تابع ()head عمل می‌کند.

۱-۴-۵ تولید داده‌ها

داده‌ها به دو صورت قابل تولید هستند، که به شرح زیر می‌باشند.

۱-۴-۵-۱ ایجاد دنباله‌های منظم از داده‌ها

دنباله منظم از اعداد صحیح، مثلاً داده‌های 1تا 30 را در نظر بگیرید.

```
> x <- 1:30
```

نتیجه دستور فوق یک آرایه‌ای با 30 عنصر است. عملگر «:»، در بین عملگرهای عددی اولویت دارد.

```
> 1:10-1
[1] 0 1 2 3 4 5 6 7 8 9
> 1:(10-1)
[1] 1 2 3 4 5 6 7 8 9
```

تابع seq() می‌تواند دنباله‌ای از اعداد حقیقی را تولید کند. به عنوان مثال:

```
> seq(1, 5, 0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

در این تابع، اولین عدد شروع دنباله، دومین عدد خاتمه دنباله و سومین عدد میزان افزایش را نشان می‌دهد. هم‌چنین می‌توان نوشت.

```
> seq(from=1, to=5, length=9)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

در اینجا فاصله بین 1 تا 5 به 9 قسمت مساوی تقسیم شده است.
با تابع c() می‌توان اعداد مورد نظر را کنار هم قرار داد و به صورت یک دنباله در آورد. به c() تابع combine نیز می‌گویند.

```
> c(1, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

تابع دیگری تحت عنوان rep() وجود دارد که اولین آرگومان آن بردار و دومین آرگومان آن تعداد تکرار عناصر آرگومان اول است. به مثال زیر توجه کنید.

```
> rep(1:4, 4)
[1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
```

آرگومان دوم نیز می‌تواند بردار باشد که تعداد عناصر آن برابر با تعداد عناصر بردار آرگومان اول است. در این حالت هر عنصر بردار آرگومان دوم، تعداد تکرار عناصر آرگومان اول را معین می‌کند. اکنون به مثال زیر توجه کنید.

```
> rep(1:4, c(2,2,2,2))
[1] 1 1 2 2 3 3 4 4
```

به مثال ساده زیر از تابع rep() توجه کنید.

```
> rep('*', 3)
[1] "*" "*" "*"
```

۱-۴-۵-۲ ایجاد دنباله‌های تصادفی از داده‌ها

این نوع دنباله توسط توزیع‌های آماری قابل تولید است. بنابراین شرح مختصری در این باره ذکر می‌گردد. زبان R شامل تعداد قابل ملاحظه‌ای از توابع جرم و چگالی احتمال است. شکل عمومی این توابع به صورت rfunc(n,

(p1, p2, ...) است که حرف r از واژه random به معنای تصادفی اخذ شده است. در آن‌ها n تعداد اعدادی است که قرار است تولید شود. حروف p1, p2, ... مقادیر پارامترهای تابع را نشان می‌دهد. به مثال زیر توجه کنید.

```
> rnorm(1)
[1] 1.358007
```

در اینجا یک عدد تصادفی نرمال استاندارد (میانگین صفر و انحراف معیار یک) تولید شده است.

۱-۵ خواندن و نوشتن داده‌ها

یکی از بخش‌های اساسی هر نرم‌افزار ورود و خروج داده‌ها در آن است. بالاخص این مطلب وقتی اهمیت خود را نشان می‌دهد که داده‌ها از یک فایل خوانده شود و یا روی یک فایل ذخیره گردد. در ادامه کار شرح این مهم خواهد آمد.

۱-۵-۱ خواندن داده‌ها از فایل

زبان R داده‌های متنی (text) را از روی فایل می‌خواند. برای این کار می‌توان از دو روش استفاده نمود.

۱-۱-۵-۱ دستور خواندن read.table

اولین دستور مورد استفاده تابع read.table() است. اگر نام فایلی data.txt باشد و مثلاً در درایو c و در پوشه‌ای به نام test ذخیره شده باشد، آنگاه دستور خواندن به صورت زیر عمل می‌کند.

```
> mydata <- read.table("c:/test/data.txt")
```

توجه داشته باشید که مسیر و یا آدرس فایل با «/» مشخص شده است، در صورتی که در سیستم عامل windowsبرای این کار از نماد «\» استفاده می‌گردد. چناچه بخواهید از نماد «\» استفاده کنید باید از دو «\\» استفاده کرد. اکنون مثال بالا به‌صورت اخیر بازنویسی می‌شود.

```
> mydata <- read.table("c:\\test\\data.txt")
```

با اجرای دستور فوق یک جدولی از داده‌ها به نام mydata تشکیل می‌شود که هر متغیر آن دارای نام است. به‌طور پیش‌فرض V1, V2, ... نامیده می‌شود و دسترسی به آن‌ها به صورت‌های mydata$V1, mydata$V2, ... و یا mydata["V1"], mydata["V2"], ... و mydata[, 1], mydata[, 2], ... می‌باشد. دستور read.table شامل خصوصیات اختیاری است که می‌توان حسب مورد از آن استفاده نمود.
اکنون به مثال زیر توجه کنید. فایلی به صورت زیر data.txt ذخیره شده است.

```
Author: John Davis
Date: 18-05-2007
```

```
Some comments...
Col1, Col2, Col3, Col4
23, 45, A, John
34, 41, B, Jimmy
12, 99, B, Patrick
```

حالا در زبان R فراخوانده می‌شود.

```
> mydata <- read.table("c:/test/data.txt", skip=3, sep=",", header=T)
> mydata
```

در تابع read.table() آرگومان skip از سه سطر اول فایل که مربوط توضیحات است عبور می‌کند و آرگومان sep ویرگول بین داده‌ها را حذف می‌نماید و بالاخره آرگومان header که از نوع منطقی است و در اینجا «T» که مخفف TRUE است ذکر شده، بنابراین اسامی داده‌ها را حفظ می‌کند. اکنون به خروجی توجه کنید.

	Col1	Col2	Col3	Col4
1	23	45	A	John
2	34	41	B	Jimmy
3	12	99	B	Patrick

در حال حاضر بسیاری از داده‌ها قبلاً در نرم‌افزار Excel ذخیره شده است. زبان R به تنهایی قادر به خواندن مستقیم فایل‌های Excel (یعنی پسوندهای `*.xls` و یا `*.xlsx`) نیست. برای خواندن چنین داده‌هایی سه راه حل وجود دارد.

۱) می‌توان داده‌ها را در Excel به فرمت `*.txt` ذخیره نمود، سپس با تابع read.table() آنها (فایل متنی `*.txt`) را خواند.

۲) می‌توان داده‌ها را در Excel به فرمت `*.csv` ذخیره نمود، سپس با تابع read.csv("نام و آدرس فایل") آنها را خواند.

۳) استفاده از package مربوطه که با استفاده از آن، زبان R مستقیماً فایل‌های Excel را می‌خواند.

در R نه به‌صورت مستقیم اما به کمک بسته نرم‌افزاری foreign که به R وصل می‌گردد، می‌توان فایل داده‌های نرم‌افزار spss را خواند. دستور خواندن به‌صورت read.spss("مسیر و نام فایل") است.

۱-۵-۱-۲ یافتن تعاملی فایل

تابع file.choose() می‌تواند فایل مورد نظر شما را به‌صورت تعاملی جستجو نموده و آن را بیابد. خروجی این تابع نام فایل و مسیر مربوط به آن را نشان می‌دهد. اکنون به مثال زیر توجه کنید.

```
> file.choose()
```

```
[1] "D:\\R_files\\data\\debit.txt"
```

اکنون اگر تابع اخیر به‌عنوان آرگومان تابع read.table() قرار گیرد، دیگر نیازی به ورود نام فایل و مسیر آن نیست، اما در هر بار اجرای برنامه باید عمل جستجو را انجام داد.

```
> read.table(file.choose())
```

تابع دیگری از همین خانواده وجود دارد که choose.files() نام دارد. این تابع می‌تواند نام و آدرس چند فایل را به‌دست دهد. با اجرای این تابع پنجره‌ای به نام Select files گشوده می‌شود. شما می‌توانید با نگه داشتن کلید Ctrl فایل‌های مورد نظر خود را انتخاب نموده و سپس کلید Open را کلیک کنید. آنگاه آدرس و نام فایل‌های انتخاب شده ظاهر می‌گردد. اکنون به مثال زیر توجه کنید.

```
> choose.files()
[1] "D:\\R_files\\data\\boise.txt" "D:\\R_files\\data\\airpass.txt"
```

همان‌طور که ملاحظه می‌شود دو فایل به نام‌های boise.txt و airpass.txt و آدرس‌های آنها خروجی تابع گفته شده هستند.

آدرس و نام فایل‌ها را می‌توان داخل یک شی مثلاً به نام x نگه داشت و در صورت لزوم به‌طور مجزا از آنها استفاده نمود. به مثال زیر توجه کنید.

```
> x <- choose.files()
> x
[1] "D:\\R_files\\data\\boise.txt" "D:\\R_files\\data\\airpass.txt"
> x[1]
[1] "D:\\R_files\\data\\boise.txt"
> x[2]
[1] "D:\\R_files\\data\\airpass.txt"
```

در انتخاب فایل‌ها می‌توان آنها را بر حسب پسوندشان فیلتر نمود. فرض کنید مثلاً فایل‌هایی مطمح نظر است که پسوند آنها txt باشد، این وظیفه را آرگومان فیلتر در تابع اخیر انجام می‌دهد. به مثال زیر توجه کنید.

```
> choose.files(filters = Filters[c("txt", "All"), ])
```

۱-۵-۱-۳ دستور خواندن scan

تابع read.table از تابع سطح پایین‌تری به نام scan() استفاده می‌کند. این تابع می‌تواند به وسیله کاربر نیز مستقیماً فراخوانده شود و در پاره‌ای از اوقات که عملکرد read.table مناسب به نظر نمی‌رسد، از آن استفاده می‌شود. شکل عمومی آن به شرح زیر است.

```
> scan(file=" ")
```

که جلوی آرگومان فایل، داخل کوتیشن، نام فایل و آدرس آن درج می‌گردد. توجه داشته باشید که اگر ساختار فایل به‌صورت یک جدول بود، تابع scan() آن را به شکل جدول نمی‌خواند و برای این که محتویات فایل را به‌صورت

جدول نگه دارید باید از تابع matrix() استفاده نمود که در قسمت ماتریس‌ها توضیح داده خواهد شد.
تابع scan() می‌تواند داده‌ها را از روی صفحه کلید بخواند. به مثال زیر توجه کنید.

```
> x <- scan()
1: 10
2: 11
3: 12
4: 13
5: 14
6: 15
7: 16
8:
Read 7 items
```

در آخرین مرحله (پس از اتمام داده‌ها) بدون ورود عدد جدید، با زدن کلید Enter می‌توان از مرحله ورود داده‌ها خارج شد. در این شکل استفاده از تابع گفته شده فقط ورود داده‌های عددی ممکن است و اگر داده‌های شما به‌صورت رشته‌ای از کاراکترها باشد تابع scan() نیازمند آرگومان what="" است. به مثال زیر توجه کنید.

```
> scan(what="")
1: 1Jan2000
2: 5Mar2001
3: 1May2002
4: 10Jun2003
5: 11Sep2004
6:
Read 5 items
[1] "1Jan2000" "5Mar2001" "1May2002" "10Jun2003" "11Sep2004"
```

در قسمت مربوط به لیست و یا فهرست به این نکته اشاره خواهد شد که چگونه این تابع هم اعداد و هم رشته‌ها را به‌طور توأم خواهد خواند.

۱-۵-۲ نوشتن داده‌ها روی فایل

دستور write.table داده‌ها را روی فایل می‌نویسد.

```
> write.table(x, file=" ")
```

که در آن x داده‌هایی است که قرار است ذخیره شود و در file=" " نام فایلی است که داده‌ها در آن ذخیره می‌شوند قرار می‌گیرد. در تابع write.table() چند آرگومان وجود دارد که درخور توجه است.

- آرگومان منطقی quote وجود دارد که به‌صورت پیش‌فرض T است و در این حالت دور اسامی ستون‌ها و شماره ردیف‌ها کوتیشن قرار می‌گیرد و در صورتی که quote=F شود کوتیشن‌ها حذف می‌گردد.
- آرگومان منطقی row.names وجود دارد که به‌صورت پیش‌فرض T (درست) است و برای هر ردیف از داده‌ها شماره قرار می‌دهد. در صورتی که row.names=F گردد شماره ردیف‌ها حذف می‌گردد.

- آرگومان دیگری به نام sep یا جداکننده وجود دارد که اگر از آن استفاده نشود ستون داده‌ها کنار هم و نامرتب در فایل نوشته می‌شود. برای جلوگیری از این کار از آرگومان sep="\t" می‌توان استفاده نمود. در این صورت ستون داده‌ها با یک فاصله معین از هم و مرتب در فایل نوشته می‌شود.

می‌توان یک فایل را به‌گونه‌ای در زبان R ذخیره نمود که قابل خواندن برای Excel باشد. برای این کار از فرمت *.csv استفاده می‌شود. صورت کلی دستور به شرح زیر است.

```
> write.csv2(x, file="*.csv")
```

توجه کنید که * نام دلخواه شما برای فایل خروجی می‌باشد. ضمناً آرگومان منطقی row.names وجود دارد که به‌صورت پیش‌فرض T (درست) است و برای هر ردیف از داده‌ها شماره قرار می‌دهد. در صورتی که row.names=F گردد شماره ردیف‌ها حذف می‌گردد. یعنی می‌توان نوشت.

```
> write.csv2(x, file="*.csv", row.names=F)
```

۱-۵-۳ محل خواندن و نوشتن با استفاده از فایل

در زبان R برای خواندن از روی یک فایل و نوشتن بر روی یک فایل، آدرس پیش فرضی وجود دارد که به آن working directory گویند. برای به‌دست آوردن آدرس آن از تابع getwd() استفاده می‌شود. به مورد زیر توجه کنید.

```
> getwd()
[1] "C:/Documents and Settings/shahin/My Documents"
```

برای تعویض این آدرس می‌توان از تابع setwd() استفاده نمود. مثلاً فرض کنید که آدرس مورد نظر شما، درایو D و پوشه R_files باشد، بنابراین دستور به‌صورت زیر است.

```
> setwd("D:/R_files")
```

اکنون می‌توانید با دستور getwd() آدرس جدید خود را کنترل کنید.

```
> getwd()
[1] "D:/R_files"
```

منتها وقتی از R خارج شوید و مجدداً R را بازکنید. آدرس جدید به آدرس پیش فرض برمی‌گردد، بنابراین آن تنظیم موقتی است. برای این که تعویض آدرس پایدار شود به‌صورت زیر عمل کنید.

- بر آیکون  کلیک راست کنید، تا پنجره‌ای ظاهر گردد. توجه داشته باشید که ارقام 2.14.0 مربوط به نسخه جاری R بوده است. بنابراین کاملاً طبیعی است که شما با نسخه دیگری از R این کار را انجام دهید. اما این امر در نتیجه عمل به هیچ‌وجه نقشی ایفا نمی‌کند.
- در پنجره حاضر گزینه properties را کلیک کنید. در جعبه مقابل عبارت Start in: آدرس مورد نظر خود را وارد کنید و سپس کلید ok را بزنید. به این ترتیب آدرس پیش فرض شما آدرس جدید خواهد شد، مگر آن که مجدداً آن را عوض کنید.

اکنون فرض کنید که می‌خواهید آدرس جدید به قوت خود باقی بماند، اما شما به یک پوشه معین که در یک مسیر خاصی قرار دارد، مراجعه مکرر دارید و مایلید به آن آدرس نیز جداگانه دسترسی مستقیم داشته باشید. برای این کار به‌صورت زیر عمل کنید.

- ابتدا وارد R شوید.
- از منوی بالای صفحه عبارت File را بزنید. سپس در پنجره حاضر گزینه Change dir ... را انتخاب کنید، آنگاه پنجره‌ای به نام Browse For Folder باز می‌گردد.
- مسیر خود را تا پوشه مورد نظر تعقیب کنید. برای اطمینان از درستی انتخاب مسیر از دستور getwd() استفاده کنید باید مسیر اختیار شده را نشان دهد.
- از منوی بالای صفحه عبارت File را بزنید. سپس گزینه Save Workspace ... را انتخاب کنید، آنگاه پنجره‌ای به نام Save image in باز می‌شود.
- پنجره حاضر همان مسیر اختیار شده را نشان می‌دهد. بدون هیچ تغییری در آن، کلید Save را بزنید.
- R را ببندید. سپس به پوشه مورد نظرتان بروید. آنگاه آیکون .RData را مشاهده می‌کنید. در اینجا پسوند RData است اما اسمی برای آن وجود ندارد. بنابراین می‌توانید یک اسم به دلخواه انتخاب نموده و با استفاده از روش Rename آن را اضافه کنید. مثلاً اگر اسم مورد نظر POT باشد نتیجه POT.RData می‌گردد.
- اکنون شما می‌توانید از آن یک shortcut روی صفحه Desktop ویندوز خود ایجاد کنید. از این shortcut وارد R شوید. اولاً در آخرین سطر نوشته‌های صفحه R جمله

[Previously saved workspace restored]

ظاهر می‌گردد. ثانیاً اگر دستور getwd() را اجرا کنید، همان آدرس مورد نظر ملاحظه می‌شود.

۱-۶ ساختار داده‌ها

در زبان R می‌توان داده‌ها را به صورت‌های زیر نگهداری نمود.

- بردار (vector)
- ماتریس (matrix)
- آرایه (array)
- داده‌های چارچوب‌دار (data frame)
- داده‌های سری زمانی (time series)
- فهرست (list)

اکنون به شرح مجزای هر یک از آن‌ها پرداخته می‌شود.

۱-۶-۱ بردارها

ساده‌ترین ساختار داده‌ها در زبان R، بردارها هستند. بردار شی است که شامل چند داده با نوع یکسان هستند، تماماً عدد و یا تماماً منطقی و . . . می‌باشند. همانطورکه قبلاً نیز ملاحظه شد می‌توان با تابع c() بردار را ساخت. به مقال زیر توجه کنید.

```
> x <- c(10,5,3,6)
> x
[1] 10 5 3 6
```

و یا

```
> y <- c(x, 0.55, x, x)
> y
[1] 10.00 5.00 3.00 6.00 0.55 10.00 5.00 3.00 6.00 10.00 5.00 3.00
[13] 6.00
```

اکنون به یک مثال کاراکتری توجه کنید.

```
> pets <- c("cat","dog","gerbil","terrapin")
> length(pets)
[1] 4
```

تابع length() نشان می‌دهد که چهار رشته وجود دارد. اما طول هر رشته به‌صورت زیر به‌دست می‌آید.

```
> pets <- c("cat","dog","gerbil","terrapin")
> nchar(pets)
[1] 3 3 6 8
```

۱-۱-۶-۱ محاسبات ریاضی روی بردارها

محاسبات روی بردارهای عددی معمولاً روی هر عنصرش انجام می‌شود. برای مثال x*x هر عنصر بردار x را مربع می‌کند.

```
> x
[1] 10 5 3 6
> z <- x*x
> z
[1] 100 25 9 36
```

می‌توان توابع را روی عناصر یک بردار اثر داد. مثلاً تابع لگارتیم را روی بردار x اعمال نمود.

```
> log(x)
[1] 2.302585 1.609438 1.098612 1.791759
```

در حالتی که دو بردار دارای طول یکسان نباشند، بردار کوتاه‌تر آنقدر تکرار می‌گردد تا به اندازه بردار طولانی‌تر شود. مثال ساده بردار و یک عدد را در نظر گیرید.

```
> sqrt(x) + 2
[1] 5.162278 4.236068 3.732051 4.449490
```

در اینجا عدد 2 چهار بار تکرار شده است تا به اندازه طول بردار x گردد و سپس هر عنصر بردار با عدد 2 جمع می‌شود. اکنون به مثالی توجه کنید که که هر دو عملوند بردار باشند.

```
> x <- c(1,2,3,4)
> y <- c(1,2,3,4,5,6)
> z <- x*y
Warning message:
In x * y : longer object length is not a multiple of shorter object length
> z
[1] 1 4 9 16 5 12
```

ایجاد زیربردار

به دو صورت می‌توان یک زیربردار را ایجاد نمود.

- شماره عناصری که باید انتخاب شود را مشخص کنید. مثال

```
> x <- c(3,11,8,15,12)
> x[c(2,4)]
[1] 11 15
```

- با استفاده از اعداد منفی (شماره عناصر) می‌توان عناصر غیر لازم را حذف نمود.

```
> x <- c(3,11,8,15,12)
> x[-c(2,3)]
[1] 3 15 12
```

بردار x به‌صورت زیر مفروض است. به مواردی که به عنوان عملیات روی آن انجام می‌شود، توجه کنید.

```
> x <- c(1,3,6,10,15)
> x[1]
[1] 1
> x[5]
[1] 15
> length(x)
[1] 5
> dim(x)
NULL
> nrow(x)
NULL
> ncol(x)
```

NULL

همان‌طور که ملاحظه می‌کنید، بردار بر خلاف ماتریس دارای ابعاد نیست. اگر بخواهید که x یک بردار ستونی گردد، باید آن را به‌صورت یک ماتریس تعریف نمود. به کُدهای زیر توجه کنید.

```
> x <- as.matrix(x)
> dim(x)
[1] 5 1
```

در سری داده‌ها ممکن است که داده یا داده‌هایی مفقود[13] شده باشند. در این صورت در زبان R آن‌ها با علامت NA نشان داده می‌شوند. نحوه‌ی عمل توابع R در مورد NAها متفاوت است، که باید به راهنمای زبان مراجعه نمود. در اینجا سه تابع به عنوان مثال ذکر می‌گردد.

۱) تابع order() داده‌های NA را به آخر سری منتقل می‌کند.

```
> x <- c(1, 20, 2, NA, 22)
> order(x)
[1] 1 3 2 5 4
> x[order(x)]
[1] 1 2 20 22 NA
```

۲) تابع sort() داده‌های NA را خودبه‌خود حذف می‌کند.

```
> x <- c(1, 20, 2, NA, 22)
> sort(x)
[1] 1 2 20 22
```

۳) تابع mean() مثل هیچ‌یک از دو تابع فوق عمل نمی‌کند و باید به‌صورت دستی (na.rm=T) کار را سرانجام داد.

```
> x <- c(1, 20, 2, NA, 22)
> mean(x)
[1] NA
> mean(x, na.rm=T)
[1] 11.25
```

البته می‌توان از راه حل زیر نیز استفاده نمود.

```
> x <- c(1,2,NA,3,4)
> y <- x[!is.na(x)]
> mean(y)
[1] 2.5
```

اکنون اگر بخواهید مثلاً NA را با مقدار 999- عوض کنید. می‌توان به‌صورت زیر عمل کرد.

```
> x[is.na(x)] <- -999
> x
[1] 1 2 -999 3 4
```

13. missing data

اکنون به مثال جالب زیر توجه کنید که مربوط به بردارها می‌شود.

```
> x <- 0:10
> x
[1] 0 1 2 3 4 5 6 7 8 9 10
> sum(x)
[1] 55
```

متغیر x اعداد 0 تا 10 را اختیار نموده و سپس مجموع آنها با تابع sum() محاسبه می‌شود. اما اگر بخواهید بخشی از مقادیر x را با هم جمع کنید. در بادی امر به نظر می‌رسد که

```
> sum(x<5)
[1] 5
```

که البته جواب درست نیست. زیرا زبان R به‌صورت زیر عمل نموده است.

```
> x<5
[1] TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
```

یعنی عبارت منطقی x<5 را در نظر گرفته و TRUEها را تبدیل به عدد 1 و FALSEها را تبدیل به عدد 0 نموده است. سپس جمع TRUEها یعنی عدد 5 محاسبه شده است. اما بالاخره برای حل مساله باید به‌صورت عمل نمود.

```
> sum(x[x<5])
[1] 10
```

تابع cumsum()

تابع cumsum() برداری را ایجاد می‌کند که طول آن برابر بردار اولیه داده‌های ورودی است و iامین عنصر آن برابر جمع i عنصر اول داده‌های ورودی است. به مثال زیر توجه کنید.

```
> cumsum(rep(2,10))
[1] 2 4 6 8 10 12 14 16 18 20
```

۱-۶-۱-۲ تابع which()

اگر شرط خاصی در یک بردار برقرار شد اندیس آن توسط تابع which() برمی‌گردد. اکنون به مثال زیر توجه کنید.

```
> vals <- c(1,3,2,68,11,13,19,8,49,4)
> my_max <- max(vals)
> which_val <- which(vals == my_max)
> cat(c("Max =", my_max, "Val#", which_val, "\n"))
Max = 68 Val# 4
```

همان‌طور که ملاحظه می‌شود با اجرای کدهای فوق عنصر حداکثر و اندیس آن به‌دست آمد.

۱-۶-۲ ماتریس‌ها

در واقع ماتریس بسط بردار است. مانند بردار تمام عناصر یک ماتریس دارای نوع داده‌های یکسان است. برای ساختن ماتریس کافی است برای عناصر تابع c() و تعداد سطر «nr و یا nrow» و تعداد ستون «nc و یا ncol» تعریف نمود. اکنون به مثال زیر توجه کنید.

```
> A <- matrix(c(1,2,3,4), nr=2, nc=2)
> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

اکنون اگر بخواهید که به یک درآیه از ماتریس دست پیدا کنید، از A[i,j] استفاده می‌شود. مثال:

```
> A[1,2]
[1] 3
```

می‌توان به سطر یا ستونی معینی از ماتریس دست یافت. مثال:

```
> A[,1]
[1] 1 2
> A[2,]
[1] 2 4
```

همانطورکه ملاحظه می‌شود A[,1] یک آرایه نیست و اگر از تابع دیمانسیون dim() استفاده شود نتیجه به‌صورت زیر است.

```
> dim(A[,1])
NULL
```

اما می‌توان از آرگومانی به نام drop استفاده نمود تا خاصیت آرایه حذف نگردد. به مثال قبل با کاربرد جدید توجه کنید.

```
> A[,1,drop=F]
     [,1]
[1,]    1
[2,]    2
> dim(A[,1,drop=F])
[1] 2 1
```

توجه داشته باشید که عناصر ماتریس در زبان R به‌صورت ستونی (پیش‌فرض) ذخیره می‌شوند. به مثال زیر توجه کنید.

```
> xx <- matrix(1:6,ncol=3) # Equivalently, enter matrix(1:6,nrow=2)
> xx
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

توجه: از علامت «#» در زبان R برای قرار دادن جملات توضیحی[۱۴] استفاده می‌شود.
اگر بخواهید که نحوه‌ی ذخیره شدن به شکل سطری باشد باید از عبارت byrow=T استفاده نمود. به مثال زیر توجه کنید.

```
> xx <- matrix(1:6, ncol=3, byrow=T)
> xx
      [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

همان‌طور که قبلاً گفته شد تابع scan() داده‌ها را به‌صورت جدول نمی‌خواند. فرض کنید که فایل داده‌هایی به نام try.txt به‌صورت زیر باشد.

```
1 2 3
4 5 6
7 8 9
```

اکنون با استفاده از تابع scan() خوانده می‌شود. خروجی به‌صورت زیر است.

```
> scan( "D:\\R_files\\try.txt")
Read 9 items
[1] 1 2 3 4 5 6 7 8 9
```

حالا از تابع ماتریس استفاده می‌گردد. مشاهده می‌شود که خروجی به‌صورت جدول حفظ خواهد شد.

```
> matrix(scan("D:\\R_files\\try.txt"), byrow=T, ncol=3)
Read 9 items
      [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

اکنون به زیرآرایه زیر توجه کنید.

```
> B <- matrix(1:6, nr=2)
> B
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> B[,2:3]
      [,1] [,2]
[1,]    3    5
[2,]    4    6
```

اگر بخواهید که یک ماتریس به صورت بردار درآید از تابع as.vector استفاده کنید. به مثال زیر توجه کنید.

```
> xx <- matrix(1:6, ncol=3)
```

۱۴. این‌گونه جملات اجرا نمی‌شوند و برای شرح دستورات و یا اعمالی که نیاز به توصیف دارند مورد استفاده قرار می‌گیرند.

```
> xx
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> x <- as.vector(xx)
> x
[1] 1 2 3 4 5 6
```

اکنون توجه کنید که چگونه می‌توان شکل یک سری از اعداد را تغییر داد. این کار توسط تابع دیمانسیون dim() صورت می‌گیرد. به مثال زیر توجه کنید.

```
> x <- 1:24
> dim(x) <- c(2,12)
> x
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
[1,]    1    3    5    7    9   11   13   15   17   19   21   23
[2,]    2    4    6    8   10   12   14   16   18   20   22   24
```

حالا به مثال زیر توجه کنید.

```
> x <- 1:24
> dim(x) <- c(3,4,2)
> x
, , 1

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

اکنون به دو مثال زیر توجه کنید که تفاوت کاربرد علامت «=» و علامت «<- » را نشان می‌دهد. مثال اول کاربرد علامت تساوی در آرگومان ماتریس است، که با خطا مواجه می‌گردد.

```
> matrix(a = 10, 5, 5)
Error in matrix(a = 10, 5, 5) : unused argument(s) (a = 10)
```

مثال دوم کاربرد علامت «<-» در آرگومان ماتریس است.

```
> matrix(a <- 10, 5, 5)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   10   10   10   10
[2,]   10   10   10   10   10
[3,]   10   10   10   10   10
[4,]   10   10   10   10   10
[5,]   10   10   10   10   10
```

که یک ماتریس 5×5 را نشان می‌دهد. البته می‌توان مقدار a را به‌طور مجزا نیز داشت.

```
> a
[1] 10
```

توابع ()rbind و ()cbind می‌توانند دو آرایه و یا دو ماتریس را بر حسب سطر و یا ستون به یکدیگر متصل نمایند. به عنوان مثال اولین ماتریس را در نظر بگیرید.

```
> m1 <- matrix(1, nr=2, nc=2)
> m1
     [,1] [,2]
[1,]    1    1
[2,]    1    1
```

دومین ماتریس مثال به صورت زیر است.

```
> m2 <- matrix(2, nr=2, nc=2)
> m2
     [,1] [,2]
[1,]    2    2
[2,]    2    2
```

اکنون تابع rbind روی ماتریس‌های m1, m2 اعمال می‌شود، آنگاه نتیجه به صورت زیر است.

```
> rbind(m1, m2)
     [,1] [,2]
[1,]    1    1
[2,]    1    1
[3,]    2    2
[4,]    2    2
```

اگر تابع cbind روی ماتریس‌های m1, m2 اعمال می‌شود، آنگاه نتیجه به صورت زیر است.

```
> cbind(m1, m2)
     [,1] [,2] [,3] [,4]
[1,]    1    1    2    2
[2,]    1    1    2    2
```

با تابع ()apply می‌توان یک تابع را روی ستون، سطر و یا سطر و ستون اعمال نمود و نیازی به برنامه‌نویسی نیست. نحو کلی به صورت apply(X, MARGIN, FUN,...) است، که در آن X ماتریس، MARGIN نشان‌دهنده سطر (۱)، ستون (۲) و یا هر دو c(1,2)، FUN تابعی است که قرار است اعمال گردد و ... آرگومان‌های اختیاری تابع است. به مثال زیر توجه کنید.

```
> x <- rnorm(10, -5, 0.1)
```

```
> y <- rnorm(10, 5, 2)
> X <- cbind(x, y)
> apply(X, 2, mean)

         x          y
-5.027833   4.617558

> apply(X, 2, sd)

         x          y
0.1023305  1.8215198
```

در زبان R، داده‌ای وجود دارد که trees نامیده می‌شود و شامل سه ستون است. تابع ()apply به عنوان مثال روی آن اعمال می‌گردد.

```
> apply(trees, 2, sum)
Girth Height Volume
410.7 2356.0 935.3
```

اکنون اگر بخواهید که فقط مثلاً جمع یک ستون (Height) را محاسبه کنید، می‌توان به دو صورت زیر عمل نمود.

```
> sum(trees[,2])
[1] 2356
```

و یا

```
> sum(trees$Height)
[1] 2356
```

اما بالاخره در هر دو صورت فوق برای محاسبه جمع، میانگین و . . . فقط از نام ستون داده‌های trees نمی‌توان استفاده نمود. برای این کار از تابعی در زبان R به نام ()attach می‌توان استفاده نمود. ابتدا نام داده را با تابع مذکور فرا می‌خوانید و سپس می‌توان با نام ستون‌ها عملیات فوق را انجام داد.

```
> attach(trees)
> sum(Height)
[1] 2356
```

برای اطلاع از تعداد و اسامی متغیرها در داخل مجموعه داده‌های trees می‌توان از تابع ()names استفاده نمود.

```
> names(trees)
[1] "Girth" "Height" "Volume"
```

توجه: در زبان R تابع ()args می‌تواند آرگومان‌های یک تابع را به شما نشان دهد. به عنوان مثال به آرگومان‌های تابع ()apply توجه کنید.

```
> args(apply)
function (X, MARGIN, FUN, ...)
NULL
```

۱-۶-۲-۱ مشاهده ساختار یک شی

در زبان R علاوه بر توابع ()names و ()args می‌توان از تابعی استفاده نمود که ()str نامیده می‌شود. نام تابع str از کلمه structure به معنای ساختار ناشی می‌گردد. این تابع می‌تواند ساختار هر شی (اعم از داده‌ها، متغیرها، توابع و ...) را نشان دهد. به مثال زیر توجه کنید.

```
> str(trees)
'data.frame':   31 obs. of  3 variables:
 $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
 $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
 $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
```

در اینجا علاوه بر تعداد و اسامی متغیرها می‌توان نوع داده‌ها، تعداد داده‌ها و قسمتی از داده‌های هر متغیر را مشاهده نمود.
و یا به عنوان مثال به ساختار تابع ()q یعنی تابع خروج، نگاه کنید.

```
> str(q)
function (save = "default", status = 0, runLast = TRUE)
```

۱-۶-۲-۲ ماتریس و عملیات آن

برای ضرب دو ماتریس از عملگر «%*%» استفاده می‌شود. برای مثال به حاصل‌ضرب دو ماتریس m1, m2 توجه کنید.

```
> rbind(m1,m2) %*% cbind(m1, m2)
     [,1] [,2] [,3] [,4]
[1,]    2    2    4    4
[2,]    2    2    4    4
[3,]    4    4    8    8
[4,]    4    4    8    8
```

و یا

```
> cbind(m1, m2) %*% rbind(m1, m2)
     [,1] [,2]
[1,]   10   10
[2,]   10   10
```

ترانهاده یک آرایه توسط تابع ()t حاصل می‌گردد. تابع ()diag برای استخراج و یا تغییر درآیه‌های قطری و یا ساختن یک ماتریس قطری به‌کار می‌رود.

```
> diag(m1)
[1] 1 1
```

```
> diag(rbind(m1, m2) %*% cbind(m1, m2))
[1] 2 2 8 8
> diag(m1) <- 10
> m1
      [,1]  [,2]
 [1,]   10     1
 [2,]    1    10
> diag(3)
      [,1]  [,2]  [,3]
 [1,]    1     0     0
 [2,]    0     1     0
 [3,]    0     0     1
> v <- c(10, 20, 30)
> diag(v)
      [,1]  [,2]  [,3]
 [1,]   10     0     0
 [2,]    0    20     0
 [3,]    0     0    30
```

می‌توان با استفاده از سطر و ستون یک ماتریس و اندیس‌های آن، ماتریس مثلث بالا و یا ماتریس مثلث پایین ایجاد نمود.

```
> A <- matrix(1:9, nc=3)
> A[row(A) > col(A)] <- 0
> A
      [,1]  [,2]  [,3]
 [1,]    1     4     7
 [2,]    0     5     8
 [3,]    0     0     9
```

زبان R توابع خاصی برای محاسبات ماتریسی دارد. دستور det() برای محاسبه دترمینان، دستور solve() برای معکوس نمودن ماتریس و دستور eigen() برای بدست آوردن مقادیر و بردارهای ویژه به‌کار می‌رود.

مثال: در اینجا ماتریس زیر معکوس می‌گردد.

```
> x <- 2
> y <- 3
> z <- 3
> t <- 4
> m <- matrix(c(2*x, y, z+1, t+1), 2)
> solve(m)
```

با اجرای کُدهای اخیر نتیجه زیر حاصل می‌گردد.

```
  [,1]  [,2]
[1,]  0.625 -0.5
[2,] -0.375  0.5
```

۱-۶-۲-۳ حل دستگاه معادلات خطی

با تابع solve() می‌توان چند معادله چند مجهول را حل نمود. برای مثال به حل دو معادله دو مجهول زیر توجه کنید.

$$\begin{cases} x_1 + 2x_2 = 1 \\ 3x_1 + 4x_2 = 1 \end{cases}$$

اگر آن را به فرم ماتریسی بنویسید به صورت زیر در می‌آید.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

اکنون به کُدهای R آن توجه کنید.

```
> A <- matrix(c(1, 3, 2, 4), ncol = 2)
> b <- c(1, 1)
> solve(A, b)
[1] -1 1
```

۱-۶-۲-۴ حل معادله چندجمله‌ای

برای حل این نوع معادلات از تابع polyroot() استفاده می‌شود. ای تابع یک آرگومان دارد که z است و آن عبارتست از بردار ضرایب چندجمله‌ای که به‌صورت افزایش توان‌های آن تنظیم می‌گردد. اکنون به رابطه زیر توجه کنید.

$$p(x) = z_1 + z_2 \times x + z_3 \times x^2 + \cdots + z_n \times x^{n-1}$$

ضرایب چندجمله‌ای به‌صورت بردار $z[1:n]$ است. حالا به چند مثال توجه کنید.
حل معادله درجه دوم $x^2 - 0.3x - 1.8 = 0$

```
> polyroot(c(-1.8,-0.3,1))
[1] 1.5+0i -1.2+0i
```

حل معادله‌درجه دوم $x^2 - 4 = 0$ که ضریب درجه اول آن برابر صفر است.

```
> polyroot(c(-4,0,1))
[1] 2+0i -2+0i
```

حل معادله درجه سوم $x^3 + 6x^2 + 11x + 6 = 0$

```
> polyroot(c(6,11,6,1))
[1] -1+0i -2-0i -3+0i
```

همان‌طورکه در مثال‌های اخیر ملاحظه شد نمایش ریشه معادلات به‌صورت اعداد مختلط است. اما در مواردی که ریشه‌ها صحیح و یا حقیقی هستند ضریب عدد موهومی i برابر با صفر است.
بالاخره به حل معادله درجه سوم $x^3 - x^2 + 1.5x - 1.5 = 0$ توجه کنید.

```
> polyroot(c(-1.5,1.5,-1,1))
[1] 1+0.000000i 0+1.224745i 0-1.224745i
```

۱-۶-۳ داده‌های چارچوب‌دار[15]

در واقع این نوع داده‌ها بسط ماتریس است. داده‌های چارچوب‌دار دارای ستون‌های با نوع داده‌های مختلف است و مناسب‌ترین ساختار داده‌ها در تجزیه و تحلیل در R می‌باشد. در واقع، اکثر روال‌های آماری در زبان R نیازمند داده‌های ورودی از این دست است. اکنون به مثال زیر توجه کنید.

```
> mtcars
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

این داده‌ها شامل اطلاعات خودروهای مختلف است. در جدول بالا هر سطر نماینده یک خودرو و ستون‌ها نمایش‌دهنده متغیرها است. در این مثال متغیر carb عدد کاربراتور را نشان می‌دهد.
شما می‌توانید تصور کنید که داده‌های چارچوب‌دار مثل یک صفحه گسترده[16] است. هر ستون آن یک بردار است. داخل هر بردار تمام عناصر نوع یکسان دارند. اما به‌هر روی بردارهای متفاوت دارای داده‌هایی با نوع‌های گوناگون هستند. تمام بردارها در این ساختار دارای طول برابر می‌باشند.

۱-۳-۶-۱ خواص داده‌های چارچوب‌دار

این نوع داده‌ها می‌توانند دارای خواص اسامی و نام سطرها باشند. خاصیت اسامی شامل اسامی ستون‌ها و خاصیت نام سطرها نمایش اسامی سطرها می‌باشند. به مثال زیر توجه کنید.

```
> rownames(mtcars)[1:5]
[1] "Mazda RX4"    "Mazda RX4 Wag"    "Datsun 710"
[4] "Hornet 4 Drive"    "Hornet Sportabout"
```

و یا

```
> names(mtcars)
[1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
[11] "carb"
```

15. data frame 16. spreadsheet

۱-۶-۳-۲ ایجاد داده‌های چارچوب‌دار

برای ایجاد این نوع داده‌ها راه‌های مختلفی وجود دارد. یکی از آن‌ها خواندن داده‌ها از طریق یک فایل است. روش دیگر استفاده از تابع data.frame() می‌باشد. به مثال‌های زیر توجه کنید.

```
> my.logical <- sample(c(T,F), size = 5, replace = T)
> my.numeric <- rnorm(5)
> my.df <- data.frame(my.logical,my.numeric)
> my.df
```

```
  my.logical  my.numeric
1      FALSE -0.05643126
2      FALSE  0.21557811
3      FALSE -0.04023551
4       TRUE -0.50746280
5      FALSE -1.63608548
```

و یا

```
> test <- matrix(rnorm(21),7,3)
> test <- data.frame(test)
> test
```

```
          X1         X2          X3
1 -0.9247492  0.18698082 -1.39943435
2 -0.6256957  0.86310566  0.91401401
3 -0.3338215 -0.74399557  0.08866272
4  0.7411739 -0.61656031  0.18208417
5  0.3436381 -0.64038437 -0.76397593
6 -1.0379184 -0.49068270  0.21339323
7 -0.5624376 -0.01380767 -1.69015277
```

اکنون از خاصیت اسامی (در مثال بالا) استفاده کنید.

```
> names(test)
[1] "X1" "X2" "X3"
```

زبان R به‌صورت خودکار اسامی ستون‌ها را X1, X2, X3 نامیده است. شما می‌توانید اسامی دلخواه خود را قرار دهید.

```
> names(test) <- c("Price", "Length", "Income")
> row.names(test) <- c("Ali","Abas","Hamid","Saeed","Amir","Mehdi","Majid")
> test
```

	Price	Length	Income
Ali	-0.9247492	0.18698082	-1.39943435
Abas	-0.6256957	0.86310566	0.91401401
Hamid	-0.3338215	-0.74399557	0.08866272
Saeed	0.7411739	-0.61656031	0.18208417
Amir	0.3436381	-0.64038437	-0.76397593
Mehdi	-1.0379184	-0.49068270	0.21339323
Majid	-0.5624376	-0.01380767	-1.69015277

همان‌طور که قبلاً ملاحظه شد عملیات متفاوتی را می‌توان با ماتریس‌ها انجام داد که با داده‌های چارچوب‌دار امکان آن وجود ندارد. برای تبدیل داده‌های چارچوب‌دار به ماتریس، می‌توان از تابع as.matrix() استفاده نمود. در زبان R می‌توان داده‌های چارچوب‌دار را به صورت یک صفحه گسترده ملاحظه نمود، و در صورت لزوم آنها را ویرایش کرد. برای این کار از توابع fix() و یا edit() استفاده می‌شود.

۴-۶-۱ تابع factor()

بعضی از متغیرها به صورت دسته‌ای یا مقوله‌ای بیان می‌شوند که آنها را در زبان R با factor نشان می‌دهند. این متغیرها به دو صورت بیان می‌شوند: غیر مرتب و مرتب.
به عنوان مثال انواع خاک‌ها را در نظر بگیرید، مثلاً شنی، ماسه‌ای، رسی و . . . که فاکتور غیر مرتب هستند. اما اگر خاک‌ها به ضعیف، قوی و خیلی قوی تقسیم کنید، این فاکتور مرتب است. همانطورکه ملاحظه می‌کنید در قسمت اخیر یک ترتیب طبیعی احساس می‌شود که در بخش اول وجود ندارد. اکنون به کُدهای زیر توجه کنید.

```
> soil.types <- c("clay","loam","sand","loam","clay")
> soil.types <- factor(soil.types)
> soil.types
[1] clay loam sand loam clay
Levels: clay loam sand
```

در قسمت levels سه نوع خاک و یا سه سطح خاک را نشان داده است. حالا به مثال زیر توجه کنید که به صورت مرتب است.

```
> soil.degrees <- c("weak","average","stronge","weak","stronge")
> soil.degrees <- ordered(soil.degrees,levels=c("weak","average","stronge"))
> soil.degrees
[1] weak average stronge weak stronge
Levels: weak < average < stronge
```

سطر آخر ترتیب سطوح را نشان می‌دهد. که می‌توان آنها به صورت عددی نیز نشان داد.

```
> soil.numeric <- as.double(soil.degrees)
> soil.numeric
[1] 1 2 3 1 3
```

۱-۶-۵ موجودیت‌های سری زمانی

در زبان R شیء سری‌های زمانی با تابع ts() ایجاد می‌گردد. دو مؤلفه در آن‌ها وجود دارد.

- داده‌ها، بردار یا ماتریسی از داده‌های عددی است که هر ستون یک سری زمانی مجزا را تشکیل می‌دهد.
- تاریخ داده‌ها، فواصل مساوی تاریخی است.

اکنون به مثال زیر توجه کنید.

```
> my.ts <- ts(matrix(rnorm(30), ncol = 2), start = c(1987), freq = 12)
> my.ts
```

		Series 1	Series 2
Jan	1987	-0.39579521	-0.9026295
Feb	1987	-0.36649473	0.2915367
Mar	1987	-0.40674973	-1.8566028
Apr	1987	0.32428545	-0.3488452
May	1987	0.55310221	-1.6910047
Jun	1987	-1.03032817	1.3434787
Jul	1987	0.70500090	1.4546442
Aug	1987	-1.47332244	1.6205861
Sep	1987	0.58314662	0.6378932
Oct	1987	1.55713060	-1.9681021
Nov	1987	0.74596283	0.8186411
Dec	1987	0.16404647	-1.1321248
Jan	1988	-0.03516499	-0.1627264
Feb	1988	-0.18482688	-1.2735001
Mar	1988	-1.16434845	-1.0717075

۱-۶-۶ فهرست‌ها[17]

لیست و یا فهرست شبیه بردار است. اما هر عنصر یک فهرست می‌تواند شی باشد که شامل هر نوع و هر ساختاری است. در نتیجه یک فهرست خود می‌تواند شامل فهرست دیگری باشد، بنابراین می‌توان برای ساختارهای مختلف داده‌ها از آن استفاده نمود. لیست و یا فهرست غالباً برای روال‌های خروجی آماری در زبان R به‌کار می‌رود. موجودیت خروجی غالباً شامل مجموعه‌ای از برآورد پارامترها، باقیمانده‌ها، مقادیر پیش‌بینی شده و غیره می‌باشد. برای مثال به خروجی تابع lsfit() توجه کنید، که ساده‌ترین فرم تابع حداقل مربعات برای رگرسیون است.

```
> x <- 1:5
> y <- x + rnorm(5,0,0.25)
> z <- lsfit(x,y)
> z
```

Intercept	X
-0.1150539	0.9848682

17. Lists

[1] 0.259665308 -0.339443370 -0.001599916 -0.017131291 0.098509268

[1] TRUE

در این مثال مقدار خروجی lsfit(x,y) به z نسبت داده شده است. این یک فهرست است که اولین مؤلفه آن بردار شیب و عرض از مبداء است. مؤلفه دوم برداری از باقیمانده‌هاست. بردار سوم با طول واحد نشان می‌دهد که آیا از عرض از مبداء استفاده شده است یا خیر؟
عناصر یک فهرست به روش‌های مختلف قابل استخراج است:

- شماره مؤلفه: قبلاً برای دسترسی به عناصر یک آرایه از یک کروشه باز و بسته [ ] استفاده می‌شود. اما برای دسترسی به مؤلفه‌های یک فهرست و یا لیست از دو کروشه تو درتو [[ ]] استفاده می‌گردد. به‌عنوان مثال z[[1]] اولین مؤلفه z را نشان می‌دهد.
- نام مؤلفه: برای دسترسی می‌توان از نام مؤلفه نیز استفاده نمود. برای این کار اول نام فهرست و یا لیست آورده می‌شود، آنگاه علامت $ و بالاخره نام مؤلفه مورد نظر قید می‌گردد. به‌عنوان مثال z$name مؤلفه‌ای از z را نشان می‌دهد که در قسمت نام به‌کار رفته است.

برای به‌کار بردن نام می‌توان از مختصر آن نیز استفاده نمود. مثلاً بجای z$residuals می‌توان از z$r بهره گرفت.

```
> test <- z$r
> test
[1] 0.259665308 -0.339443370 -0.001599916 -0.017131291 0.098509268
> z$r[4] # fourth element of the residuals
[1] -0.01713129
```

۱-۶-۶-۱ ایجاد فهرست

برای ایجاد فهرست باید از تابع list() استفاده نمود. اسامی مؤلفه‌های فهرست و محتوای مؤلفه‌های فهرست آرگومان‌های تابع لیست هستند.

```
> x1 <- 1:5
> x2 <- c(T,T,F,F,T)
> y <- list(numbers = x1, wrong = x2)
> y
$numbers
[1] 1 2 3 4 5
 $wrong
[1] TRUE TRUE FALSE FALSE TRUE
```

همان‌طور که ملاحظه می‌کنید سمت چپ عملگر «=» نام مؤلفه است و طرف راست آن یک شیء زبان R است. مرتبه مؤلفه به ترتیب قرارگرفتن آن‌ها از چپ به راست است. در مثال بالا شیء منطقی wrong مؤلفه دوم y است.

```
> y[[2]]
[1] TRUE TRUE FALSE FALSE TRUE
```

تابع names() می‌تواند اسامی مؤلفه‌های فهرست را استخراج کند. می‌توان اسامی فهرست را تغییر داد.

```
> names(y)
[1] "numbers" "wrong"
> names(y) <- c("lots", "valid")
> names(y)
[1] "lots" "valid"
```

به فهرست قبلی می‌توان مؤلفه‌های دیگری افزود.

```
> y[[3]] <- 1:30
> y$test <- "hello"
> y
$lots
[1] 1 2 3 4 5

$valid
[1] TRUE TRUE FALSE FALSE TRUE

[[3]]
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30

$test
[1] "hello"
```

اکنون به تفاوت بین یک کروشه و دو کروشه توجه کنید.

```
> y[1]
$lots
[1] 1 2 3 4 5

> y[[1]]
[1] 1 2 3 4 5
```

وقتی از یک کروشه استفاده می‌شود، مؤلفه‌ای که حاصل می‌گردد خود یک فهرست است. اما هنگامی که از دو کروشه استفاده می‌شود فقط خود مؤلفه به‌دست می‌آید.
اکنون به مثال زیر توجه کنید که یک لیست را نشان می‌دهد.

```
> x <- list(1, c(2, 3), c(4, 5, 6))
> x
```

محتوای x به‌صورت زیر است.

```
[[1]]
```

```
[1] 1


[[2]]
[1] 2 3


[[3]]
[1] 4 5 6
```

مثال:

```
> x1 <- 1:5
> x2 <- c(T,T,F,F,T)
> x3 <- matrix(c(0,2,3,5,7,9), nr=2)
> y <- list(numbers=x1, wrong=x2, mat=x3)
> y[[3]]
```

پاسخ کُد بالا به‌صورت زیر است.

```
     [,1] [,2] [,3]
[1,]    0    3    7
[2,]    2    5    9
```

اکنون اگر بخواهید از مؤلفه سوم که ماتریس است ستون دوم آن را داشته باشید، فقط دستور آخر را به‌صورت زیر تغییر دهید.

```
> y[[3]][,2]
[1] 3 5
```

یا می‌توان دستور اخیر را به‌صورت زیر نوشت.

```
> y[[3]][,2, drop=F]
```

پاسخ کُد بالا به‌صورت زیر است.

```
     [,1]
[1,]    3
[2,]    5
```

یا می‌توان از دستور y$mat[,2] و یا می‌توان از دستور y$mat[,2, drop=F] استفاده نمود که همان نتایج قبلی حاصل می‌گردد.

اکنون با تابع ()unlist می‌توان یک لیست را به یک بردار تبدیل نمود.

```
> x <- list(1, c(2, 3), c(4, 5, 6))
> unlist(x)
[1] 1 2 3 4 5 6
```

در قسمت خواندن ملاحظه شد که تابع scan() به‌صورت مجزا می‌تواند اعداد و رشته‌ای از کاراکترها را بخواند. اکنون به کمک لیست و یا فهرست سعی می‌شود که هر دو آنها با هم خوانده شود. به مثال زیر توجه کنید.

```
> names <- scan(what=list(a=0, b="", c=0))
1: 1 dog 3
2: 2 cat 5
3: 3 duck 7
4:
Read 3 records
```

همان‌طور که ملاحظه می‌شود سه رکورد و یا سه سطر خوانده شده است. اکنون باید محتوی سطرها را مشاهده نمود.

```
> names
```

نتیجه به‌صورت زیر است.

```
$a
[1] 1 2 3

$b
[1] "dog"  "cat"  "duck"

$c
[1] 3 5 7
```

۱-۶-۷ نام‌گذاری سطر و ستون در یک جدول

اگر سطرها و ستون‌های داده‌های مورد نظر فاقد نام باشند و یا دارای نام دلخواه نباشد می‌توان آن‌ها را نام‌گذاری نمود. به مثال زیر توجه کنید.

```
> x <- 1:3
> names(x)
NULL
> names(x) <- c("a", "b", "c")
> x
 a  b  c
 1  2  3
> names(x)
[1] "a" "b" "c"
> names(x) <- NULL
> x
```

```
[1] 1 2 3
```

برای ماتریس‌ها، ستون‌ها و سطرها را توسط توابع rownames() و colnames() قابل نام‌گذاری هستند. این کار توسط دستور dimnames() انجام می‌شود.

```
> x <- matrix(1:4, 2)
> rownames(x) <- c("a","b")
> colnames(x) <- c("c","d")
> x
  c d
a 1 3
b 2 4
> dimnames(x)
[[1]]
[1] "a" "b"

[[2]]
[1] "c" "d"
```

البته مورد فوق توسط تابع dimnames() نیز انجام می‌شود.

```
> x <- matrix(1:4, 2)
dimnames(x) <- list(c("a","b"), c("c","d"))
  c d
a 1 3
b 2 4
```

حتی در زبان R می‌توان دو سطح از برچسب را ارایه نمود. به مثال فوق که دارای دو برچسب خواهد شد توجه کنید.

```
> x <- matrix(1:4, 2)
dimnames(x) <- list(rows = c("a","b"), cols = c("c","d"))
     cols
rows c d
   a 1 3
   b 2 4
```

۷-۱ کار با رشته‌ای از کاراکترها

اصولاً نرم‌افزار R برای انجام محاسبات و عملیات عددی طراحی شده است، اما بخوبی از عهده انجام عملیات با رشته‌ها نیز برمی‌آید. طبعاً برای انجام امر گفته شده از یک سری توابع استفاده می‌شود که شرح آن ذیلاً می‌آید.

nchar(): برای به‌دست آوردن تعداد عناصر یک بردار از تابع length() استفاده می‌شود. ولی برای تعداد کاراکترهای یک رشته از تابع nchar() استفاده می‌شود. اکنون به مثال زیر توجه کنید.

تابع length() نشان می‌دهد که چهار رشته وجود دارد. اما طول هر رشته به‌صورت زیر به‌دست می‌آید.

```
> pets <- c("cat","dog","gerbil","terrapin")
```

```
> nchar(pets)
[1] 3 3 6 8
```

cat(): این تابع شبیه تابع print() عمل می‌کند اما در واقع رشته‌ها را به هم متصل می‌گرداند. اکنون به مثال زیر توجه کنید.

```
> x <- 7
> y <- 10
> cat("x should be greater than y, but x=", x, "and y=", y, "\n")
x should be greater than y, but x= 7 and y= 10
```

"\n" برای تعویض خط و رفتن به خط جدید است.

paste(): برای کنترل بیشتر روی اتصال رشته‌ها به هم، از تابعی به نام paste() استفاده می‌شود. این تابع به تعداد نامحدود اسکالر می‌پذیرد و با بقیه رشته‌ها عمل اتصال را انجام می‌دهد و به‌صورت پیش‌فرض یک جای خالی بین رشته‌ها می‌گذارد. مثلاً

```
> paste("one",2,"three",4,"five")
[1] "one 2 three 4 five"
```

اما اگر بخواهید که غیر جای خالی کاراکتر دیگری بین رشته‌ها قرارگیرد از آرگومان sep در تابع paste() استفاده می‌شود. مثلاً

```
> paste("one",2,"three",4,"five", sep="_")
[1] "one_2_three_4_five"
```

اما اگر داده‌ها به‌صورت یک بردار باشند آرگومان sep نمی‌تواند کاراکتر مورد نظر را بین رشته‌ها قرار دهد. اکنون به مثال زیر توجه کنید.

```
> paste(c("one","two","three","four"), sep=" ")
```

با اجرای دستور اخیر، ملاحظه می‌شود که یک جای خالی بین رشته‌ها منظور نشده است.

```
[1] "one"   "two"   "three" "four"
```

در این حالت از آرگومان collapse استفاده می‌شود. مجدداً دستور اخیر با آرگومان جدید اجرا می‌گردد.

```
> paste(c("one","two","three","four"), collapse=" ")
[1] "one two three four"
```

اگر در تابع paste() از چند آرگومان استفاده شود. می‌توان از آرگومان‌های sep و collapse تواماً استفاده نمود. اکنون به مثال زیر توجه کنید.

```
> paste(c("X","Y"),1:5, sep="_", collapse="|")
[1] "X_1|Y_2|X_3|Y_4|X_5"
```

۱-۷-۱ زیررشته

بخشی از یک رشته را زیررشته نامند. در R برای ساختن زیررشته از توابع substr() و substring() استفاده می‌شود. البته تابع اخیر کاربرد بیشتری نسبت به تابع substr() دارد. اکنون به مثال زیر توجه کنید.

```
> substr("Tehran", 2, 5)
[1] "ehra"
```

همان‌طور که ملاحظه می‌شود مقدار 2 شروع زیررشته و مقدار 5 خاتمه زیررشته را معین می‌کند. اکنون به یک مثال مفصل‌تر توجه کنید.

```
> province <- "Abadan"
> Len <- nchar(province)
> Len
[1] 6
> Ltr <- substring(province, 1:6, 1:6)
> Ltr
[1] "A" "b" "a" "d" "a" "n"
> which(Ltr == "a")
[1] 3 5
```

توضیح: تابع nchar() طول رشته را برمی‌گرداند. در تابع substring() از کاراکتر اول تا کاراکتر ششم از یکدیگر جدا می‌شوند. در تابع which() محل قرارگرفتن کاراکتر a در رشته Abadan مشخص می‌گردد.
اکنون به مثال زیر توجه کنید.

```
> mystring <- "karoon karkheh"
> substring(mystring,4, 7) <- "aje"
> mystring
[1] "karaje karkheh"
```

ملاحظه می‌شود که به‌جای کاراکترهای oon کاراکترهای aje جایگزین می‌گردد.

۱-۸ تاریخ و زمان در زبان R

اندازه‌گیری زمان امری کاملاً شخصی است. زیرا سالیان متوالی از روزها و ماه‌های مختلف شروع می‌شود. ماه‌های دارای روزهای مختلف هستند. سال کبیسه وجود دارد. ترتیب نوشتن روز و ماه می‌تواند متفاوت باشد. مثلاً 3/4/2006 به معنای روز چهارم ماه آوریل است.
با توجه به مراتب فوق کار با عناصر تاریخ و زمان امر نسبتاً دشواری است. خوشبختانه در زبان R سیستم استواری برای کار با این عوامل پیچیده تعبیه شده است.
اولین تابعی که تاریخ و زمان را به‌دست می‌دهد تابع Sys.time() است که پاسخ آن کاملاً سلسله مراتبی است. بدین معنی که از چپ به راست بزرگترین مقیاس یعنی سال، ماه و بالاخره روز را که با علامت خط تیره از هم جدا شده‌اند را به‌دست می‌دهد. سپس یک جای خالی و در پی آن نوبت به زمان می‌رسد که به ترتیب ساعت،

دقیقه و ثانیه را به‌دست می‌دهد که با علامت «:» از هم جدا شده‌اند. در آخرین مرحله یک رشته از کاراکترها را که زمان محلی را نشان می‌دهد، نوشته می‌شود.

```
> Sys.time()
[1] "2012-04-12 16:50:49 IRDT"
```

برای استخراج تاریخ از تابع Sys.time() از تابع زیر رشته استفاده می‌گردد.

```
> substr(as.character(Sys.time()),1,10)
[1] "2012-04-12"
```

البته برای نشان فقط تاریخ روز می‌توان از تابع Sys.Date() استفاده نمود.

```
> Sys.Date()
[1] "2012-04-12"
```

۱-۸-۱ محاسبات با تاریخ و زمان

برای ساختن شی تاریخ و زمان از تابع as.Date استفاده می‌گردد. به عبارت دیگر این تابع یک بردار کاراکتری را به یک بردار تاریخی تبدیل می‌کند. اولین آرگومان آن یک رشته به‌صورت "year-month-day" مانند "2007-09-24" است. اگر تاریخ و زمان در نظر باشد، رشته فوق به‌صورت "year-month-day hour:minutes:seconds" است. دومین آرگومان تابع گفته شده format است که تبدیل تاریخ و زمان را به‌صورت رشته‌های متفاوت بیان می‌کند.

با تاریخ و زمان می‌توان محاسبات مختلفی انجام داد. این محاسبات را می‌توان به موارد زیر تقسیم نمود.

- time + number
- time - number
- time - time
- time1 'logical operation' time2

اکنون به چند مثال زیر توجه کنید.

```
> as.Date("2007-10-18", format = "%Y-%m-%d")
[1] "2007-10-18"
```

توجه داشته باشید که علامت درصد «%» باید با علامت‌های سال، ماه و روز همراه شود.

```
> as.Date("02/27/92", "%m/%d/%y")
[1] "1992-02-27"
```

همان‌طور که مشاهده می‌شود علامت y برای سال دو رقمی است و علامت Y برای سال چهار رقمی است. ضمناً اگر در تاریخ از جداکننده «-» استفاده می‌شود در فرمت هم علامت همان است. اما اگر در تاریخ از علامت جداکننده «/» استفاده می‌گردد فرمت هم از همان علامت پیروی می‌کند.

```
>as.Date("2007oct18", format = "%Y%b%d")
[1] "2007-10-18"
```

علامت اختصاری b نشان‌دهنده نام ماه بر حسب حروف (سه حرف اول نام ماه) است. ضمناً چون بین سال، ماه و روز جداکننده‌ای وجود ندارد، در فرمت نیز علامتی قید نشده است.

توجه: این فرمت و نظایر آنها وقتی کار می‌کند که در Control Panel سیستم عامل Windows در بخش Regional and Language Options، در زبانه Regional Options، در کادر Standard and formats، در جعبه زبان، زبان انگلیسی درج شده باشد. در غیر این‌صورت پیغام NA دریافت خواهید نمود.

```
> as.Date("October 18, 2007", format = "%B %d, %Y")
[1] "2007-10-18"
```

علامت اختصاری B نشان‌دهنده نام کامل ماه بر حسب حروف است. همچنین علامت جداکننده «,» در فرمت آمده است.

```
> x <- seq.Date(from = as.Date("2007-10-18"), to = as.Date("2007-10-30"),
+ by = "3 days")
> x
[1] "2007-10-18" "2007-10-21" "2007-10-24" "2007-10-27" "2007-10-30"

> x + 10
[1] "2007-10-28" "2007-10-31" "2007-11-03" "2007-11-06" "2007-11-09"

> x - as.Date(c("2006-01-10", "2007-08-15", "2005-06-24", "2004-12-30",
+ "2005-04-05"))
Time differences in days
[1] 646 67 852 1031 938

> x > as.Date("2007-10-21")
[1] FALSE FALSE TRUE TRUE TRUE
```

و بالاخره آخرین مثال در این قسمت به‌صورت زیر است.

```
> format(Sys.Date(), "%a %b %d")
[1] "Fri Apr 20"
```

علامت اختصاری a نشان‌دهنده نام روز، بر حسب سه حرف آن است.

```
> as.Date(format(Sys.Date(), "%a %b %d"), "%a %b %d")
[1] "2012-04-20"
```

اکنون با آن می‌توان عملیات انجام داد. به مورد زیر توجه کنید.

```
> as.Date(format(Sys.Date(), "%a %b %d"), "%a %b %d") - 7
[1] "2012-04-13"
```

۱-۸-۱-۱ تاریخ و زمان در Excel

در این بخش ارتباط تاریخ و زمان در R و نرم‌افزار Excel تا حدودی بررسی می‌گردد. تاریخ اولین روز در Excel برابر 1900-01-01 است و قبل از این تاریخ موجودیتی نیست. بنابراین عملاً به تاریخ قبل از آن دسترسی وجود ندارد. اما Rدر بند این قید نیست. به مثال زیر توجه کنید.

```
> as.Date(35981, origin="1899-12-30")
[1] "1998-07-05"
```

عدد 35981 تعداد روزهایی است که به تاریخ اصلی افزوده می‌شود.
اکنون به مثال زیر توجه کنید. ابتدا فایل ورودی خوانده می‌شود و سپس قسمتی از آن را مشاهده خواهید نمود.

```
> # STEP 1: SETUP - Source File
> link= "D:\\R_files
Date.txt"
> # STEP 2: READ DATA
> my_data <- read.table(link,
+ sep = ",", dec=".", skip = 1,
+ row.names = NULL, header = FALSE,
+ col.names = c("char_date", "T_anom", "Enso_f"))
> head(my_data)
```

	char_date	T_anom	Enso_f
1	5/15/1951	0.02	2
2	6/15/1951	-0.03	2
3	7/15/1951	-0.01	3
4	8/15/1951	0.12	3
5	9/15/1951	0.09	3
6	10/15/1951	0.14	3

ملاحظه می‌شود که تاریخ داده‌ها به فرمت Excel است. حالا به دنباله کُدها توجه کنید. سپس قسمتی از خروجی را مشاهده کنید.

```
# STEP 3:Convert character date to Date, then get month value
r_date <- as.Date(my_data$char_date, "%m/%d/%Y")
r_mo <- months(r_date)
# new data.frame - add r_mo vector
my_data_1 <- data.frame(my_data, r_date)
head(my_data_1)
```

	char_date	T_anom	Enso_f	r_date
1	5/15/1951	0.02	2	1951-05-15
2	6/15/1951	-0.03	2	1951-06-15
3	7/15/1951	-0.01	3	1951-07-15
4	8/15/1951	0.12	3	1951-08-15
5	9/15/1951	0.09	3	1951-09-15
6	10/15/1951	0.14	3	1951-10-15

ستون chart_date مربوط به تاریخ Excel است و ستون آخر یعنی r_date ستون تبدیل یافته به سیستم R است.

۹-۱ برنامه‌نویسی با زبان R

برتری زبان R بر نرم‌افزارهای مشابه خود این است که می‌توان دستورات ساده‌ای را در آن نوشت و اجرا نمود. یعنی ویژگی‌های یک زبان برنامه‌نویسی در آن مستتر است، اما دارای خصوصیات ویژه‌ای است که برنامه‌نویسی را برای افراد غیرمتخصص آسان‌تر می‌سازد.

مانند سایر زبان‌ها R دارای ساختارکنترلی است که بی شباهت به فرامین زبان C نیست.

۱-۹-۱ شرط

در پاره‌ای از مواقع وقتی یک دستور باید اجرا شود که شرطی برقرارگردد و اگر این شرط برقرار نشود، دستورات دیگری اجرا خواهد شد. به شکل عمومی دستور شرط توجه کنید.

```
if (logical exp.)
{
   then do this
} else {
   do this
}
```

در مثال زیر اگر مقدار متغیر x زوج باشد، باقی‌مانده تقسیم زوج می‌شود و حرف T چاپ می‌شود.

```
> x <- 8
if (x %% 2 == 0) print("T")
[1] "T"
```

در اینجا می‌توان از دستورکارا ولی کوتاهی استفاده نمود که به صورت تابع ifelse() نشان داده می‌شود. شرح کلی تابع به صورت زیر است.

```
ifelse(test, yes, no)
```

آرگومان‌ها:

test: یک عبارت منطقی است.

yes: اگر عبارت منطقی درست بود، آنگاه yes اجرا می‌شود.
no: اگر عبارت منطقی نادرست بود، آنگاه no اجرا می‌گردد.
اکنون به مثال زیر توجه کنید.

```
> x <- c(2:-2)
> sqrt(ifelse(x >= 0, x, NA))
[1] 1.414 1.000 0.000 NA NA
```

۱-۹-۲ حلقه

حلقه برای عملیاتی است که چندین بار تکرار می‌شود. دستور کلی آن به شرح زیر است.

```
for (i in start:finish) {
execute task
}
```

البته همان‌طور که ملاحظه می‌شود انتهای حلقه مشخص است. به عبارت دیگر از قبل معلوم است که حلقه مورد نظر چند بار تکرار می‌شود.
توجه: در حلقه، آرایه‌ها بر خلاف معمول دارای اندیس می‌شوند. لذا متغیرهای مورد استفاده در آن باید قبل از شروع حلقه به نحوی مشخص و یا دارای مقدار شوند وگرنه برنامه با خطا مواجه می‌شود. برای این کار از توابعی مانند vector() (عددی، کاراکتر و منطقی)، matrix() و یا c() استفاده می‌شود. در مثال‌های زیر به کاربرد آنها دقت شود.
اکنون به مثال زیر توجه کنید.

```
> y <- vector(mode = "numeric")
> for (i in 1:10) {
+ y[i] <- i}
> y
[1] 1 2 3 4 5 6 7 8 9 10
```

مثال فوق را می‌توان به‌صورت‌های زیر نیز نوشت.

```
> y <- c()
> for(i in 1:10) {
+ y[i] <- i}
> y
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> y <- c()
> for(i in 1:10) {
+ y[i] <- i
+ print(y[i])
+ }
[1] 1
```

```
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

```
> y <- c()
> for(i in 1:10) print(y[i] <- i)
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

در صورت آخری اگر دستور print(y[i]=i) باشد، برنامه خطا خواهد داد، که این مطلب مثالی است که عملکرد دو عملگر انتساب «=» و <- در همه جا یکسان نیست.

مقدار شروع شمارنده حلقه می‌تواند از مقدار انتهایی آن بزرگتر باشد، در این صورت گام‌های بعدی در واقع 1- است. به مثال زیر توجه کنید.

```
> y <- c()
> for(i in 10:5) print(y[i] <- i)
[1] 10
[1] 9
[1] 8
[1] 7
[1] 6
[1] 5
```

واضح است که اگر گام‌های افزایش برابر 1 نبود دیگر نمی‌توان از گرامر m:n استفاده نمود. در این حالت می‌توان از تابع seq() کمک گرفت. به مثال زیر توجه کنید.

```
> y <- c()
> for(i in seq(1,5,0.5)) print(y[i] <- i)
[1] 1
[1] 1.5
[1] 2
[1] 2.5
```

```
[1] 3
[1] 3.5
[1] 4
[1] 4.5
[1] 5
```

در صورت استفاده از تابع seq() اگر مقدار شروع شمارنده حلقه از مقدار انتهایی آن بزرگتر بود، باید گام منفی را صریحاً بیان نمود زیرا در این حالت پیش‌فرضی وجود ندارد. به مثال زیر توجه کنید.

```
> y <- c()
> for(i in seq(5,1,-0.5)) print(y[i] <- i)
[1] 5
[1] 4.5
[1] 4
[1] 3.5
[1] 3
[1] 2.5
[1] 2
[1] 1.5
[1] 1
```

می‌توان حلقه‌های تو در تو داشت. به مثال زیر توجه کنید.

```
> z <- matrix(2, 4)
for(i in 1:2) {
+ for(j in 1:4) z[i,j] <- i+j}
> z
     [,1] [,2] [,3] [,4]
[1,]    2    3    4    5
[2,]    3    4    5    6
```

اما می‌توان حلقه‌هایی داشت که از قبل انتهای آنها مشخص نیست. یعنی از قبل روشن نیست که تعداد تکرار حلقه چند تا است. برای این کار از دو دستور while و repeat استفاده می‌شود.

اکنون روش نیوتن را در نظر بگیرید که روش مشهوری برای یافتن ریشه معادله جبری $f(x) = 0$ است. اگر $f'(x)$ مشتق تابع $f(x)$ باشد، آنگاه تکرار زیر به سمت ریشه معادله همگرا خواهد شد. فرض کنید که x_0 حدس اولیه باشد، بنابراین

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

این ایده مبتنی بر تقریب تیلور است.

$$f(x_n) \approx f(x_{n-1}) + (x_n - x_{n-1}) f'(x_{n-1})$$

اکنون به‌عنوان مثال تابع زیر را در نظر بگیرید.

$$f(x) = 5x^3 - 7x^2 - 40x + 100$$

آنگاه معادله بازگشتی به‌صورت زیر است.

$$x_n = x_{n-1} - \frac{5x^3 - 7x^2 - 40x + 100}{15x^2 - 14x - 40}$$

اکنون حل معادله گفته شده با دستور while به‌صورت کُدهای زیر است.

```
> x <- 0.5
> f <- 5 * x^3 - 7 * x^2 - 40 * x + 100
> tolerance <- 1e-6
> n <- 0
> while (abs(f) > tolerance) {
+ f.prime <- 15 * x^2 - 14 * x - 40
+ x <- x - f / f.prime
+ f <- 5 * x^3 - 7 * x^2 - 40 * x + 100
+ n <- n + 1
+ }
> cat("number of iterations:",n,"\n")
```

با اجرای کُدهای فوق حاصل به‌صورت زیر است.

```
number of iterations: 10
> x
[1] -3.151719
```

در دستور while تا عبارت منطقی داخل پرانتز درست است، حلقه به تکرار خود ادامه می‌دهد. در مثال فوق 10 مرتبه تکرار صورت گرفته است.

اکنون همین مثال با دستور repeat اجرا می‌گردد.

```
> x <- 0.5
> tolerance <- 1e-6
> n <- 0
> repeat {
+ f <- 5 * x^3 - 7 * x^2 - 40 * x + 100
+ if (abs(f) < tolerance ) break
+ f.prime <- 15 * x^2 - 14 * x - 40
+ x <- x - f / f.prime
+ n <- n + 1
+ }
> cat("number of iterations:",n,"\n")
```

با اجرای کُدهای فوق حاصل به‌صورت زیر است.

```
number of iterations: 10
> x
[1] -3.151719
```

دستورات داخل آکولادهای فرمان repeat آنقدر تکرار می‌شود تا عبارت منطقی داخل if نادرست گردد، آنگاه با فرمان break کنترل از حلقه خارج می‌گردد. همان‌طور که ملاحظه می‌شود تعداد تکرار برابر 10 است.

مثال: اکنون به مثال زیر توجه کنید که مثال دیگری از دستور repeat است.

```
> i <- 0
> repeat {
+ if (i > 10)
+ break
+ if (i > 2 && i < 5) {
+ i <- i + 1
+ next
+ }
+ print(i)
+ i <- i + 1
+ }
```

در حلقه اخیر دستوری به نام next وجود دارد که اگر حلقه به آن برسد بقیه فرامین آن را اجرا نمی‌کند و به ابتدای حلقه که در اینجا دستور repeat است، باز می‌گردد و گردش حلقه مجدداً آغاز می‌گردد. حاصل کُدهای بالا به‌صورت زیر است.

```
[1] 0
[1] 1
[1] 2
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

اما تا آنجا که امکان دارد، باید از به‌کار بردن شرط و حلقه خودداری نمود و از خاصیت بسیار کارای آرایه استفاده نمود. این خاصیت خود متضمن نوعی حلقه می‌باشد. مثال زیر اعداد زوج بین 1 تا 10 را بدست می‌دهد.

```
> x <- c(1:10)
> y <- x[x %% 2 == 0]
> y
[1] 2 4 6 8 10
```

توابعی نظیر «apply» لزوم استفاده از حلقه را بسیار کم می‌کند. نحوه دستور به صورت apply(X, Margin, Function) است. در آن X ماتریس است و Margin سطرها (1)، ستون‌ها (2) و یا هر دو c(1,2) هستند و Function عمل و یا تابعی است که روی داده‌ها اعمال می‌شود. در مورد توابع باید گفت که آنها می‌توانند پیش‌ساخته و یا تعریف شده کاربر باشند.

```
> x <- rnorm(10,-5,0.1)
> y <- rnorm(10,5,2)
> X <- cbind(x,y)
> apply(X, 2, mean)
         x          y
 -5.022379   4.021949
> apply(X, 2, sd)
          x           y
 0.07029347  1.73544377
```

۱-۹-۲-۱ خانواده توابع apply

همان‌طور که در بالا ملاحظه شد برای پرهیز از اعمال حلقه برای انجام یک سری از عملیات تکراری از تابعی به نام apply() استفاده شد. برای مثال در فهرست داده‌های زبان R داده‌ای به نام trees وجود داد. که جندتای اول آن به صورت زیر (کل آن 31 سطر می‌باشد.) است.

```
> head(trees,7)
```

که حاصل برابر است با:

```
 Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
4  10.5     72   16.4
5  10.7     81   18.8
6  10.8     83   19.7
7  11.0     66   15.6
```

اکنون قرار است که میانگین هر یک از ستون‌های Girth، Height و Volume محاسبه گردد.

```
> apply(trees,2,mean)
```

عدد 2 برای این به‌کار می‌رود که از ستون‌ها میانگین گرفته شود. بنابراین حاصل به‌صورت زیر است.

```
 Girth   Height   Volume
13.24839 76.00000 30.17097
```

اکنون همین کار را می‌توان با تابع ()sapply نیز انجام داد. به دستور زیر توجه کنید.

```
> sapply(trees,mean)
```

با اجرای آن همان نتیجه قبل حاصل می‌گردد. اما تابع ()sapply را می‌توان در جاهایی به‌کار گرفت (موارد پیچیده‌تر) که تابع ()apply کاربرد ندارد. بنابراین مساله را ساده‌تر (s از اول کلمه simple می‌آید.) می‌کند. به مثال زیر توجه کنید.
ابتدا یک تابع ساخته می‌شود که عمل محاسبه فاکتوریل را انجام می‌دهد. البته زبان R تابع فاکتوریل را به‌طور پیش ساخته دارد.

```
fact <- function(x) {
f <- 1
if (x < 2) return (1)
for (i in 2:x) {
f <- f*i }
f }
```

اکنون قرار است که فاکتوریل اعداد 0 تا 5 محاسبه گردد. به دستور زیر توجه کنید.

```
> sapply(0:5,fact)
```

حاصل برابر است با:

```
[1]   1   1   2   6  24 120
```

از خانواده تابع ()apply تابع دیگری وجود دارد و برای اشیایی به‌کار می‌رود که به‌صورت فهرست یا لیست (l از اول کلمه list می‌آید.) هستند. این تابع ()lapply نامیده می‌شود. اکنون به مثال زیر توجه کنید. ابتدا یک لیست ساده توسط کُدهای زیر ساخته می‌شود.

```
> a <- c("a","b","c","d")
> b <- c(1,2,3,4,4,3,2,1)
> c <- c(T,T,F)
> list.object <- list(a,b,c)
> list.object
```

حاصل برابر است با:

```
[[1]]
[1] "a" "b" "c" "d"
```

```
[[2]]
[1] 1 2 3 4 4 3 2 1

[[3]]
[1]  TRUE  TRUE FALSE
```

اکنون قرار است که تابع طول روی عناصر لیست اعمال گردد.

```
> lapply(list.object,length)
```

حاصل برابر است با:

```
[[1]]
[1] 4

[[2]]
[1] 8

[[3]]
[1] 3
```

یا برای به‌دست آوردن کلاس هر یک از عناصر به مورد زیر توجه کنید.

```
> lapply(list.object,class)
```

حاصل برابر است با:

```
[[1]]
[1] "character"

[[2]]
[1] "numeric"

[[3]]
[1] "logical"
```

اکنون اگر تابع lapply() را روی میانگین داده‌های trees اعمال کنید، همان جواب تابع sapply() را می‌دهد منتها صورت لیست در آن ظاهر می‌گردد. به مورد زیر توجه کنید.

```
> lapply(trees,mean)
```

حاصل برابر است با:

```
$Girth
[1] 13.24839

$Height
[1] 76

$Volume
[1] 30.17097
```

از خانواده تابع apply() تابع دیگری وجود دارد و برای جداول به‌کار می‌رود که در آن گروه‌بندی نیز وجود دارد. این تابع tapply() نامیده می‌شود. اکنون به مثال زیر توجه کنید.
در زبان R داده‌ای وجود دارد که mtcars نامیده می‌شود که قسمتی از این داده (8 ردیف) در زیر نمایش داده می‌شود. کل داده‌ها 32 ردیف است.

```
> data(mtcars)
> attach(mtcars)
> head(mtcars,8)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2

در جدول بالا، ستون mpg نشان‌دهنده مصرف خودروها بر حسب مایل در هر گالن است و ستون cyl مشخص کننده تعداد سیلندر خودرو است. اکنون قرار است متوسط مصرف سوخت خودروهای 4، 6 و 8 سیلندر (برای تمام ردیف‌ها) محاسبه گردد.

```
> tapply(mpg,cyl,mean)
       4        6        8
26.66364 19.74286 15.10000
```

۱-۹-۲-۲ تابع with

این تابع روی عبارات R در محیط داده‌ها عمل می‌کند. شما می‌توانید با تابع گفته شده روی توابعی نظیر tapply و یا plot که آرگومان پیش‌فرض داده ندارند عمل کنید. اگر داده‌های چارچوب‌دار قسمتی از یک بسته باشند، می‌توان داده گفته شده را مستقیماً داخل تابع with فراخواند. صورت کلی تابع عبارتست از:

```
with(data, function(...))
```

اکنون به مثال زیر توجه کنید.

```
> library(MASS)
> data(bacteria)
> with(bacteria, tapply((y=="n"), trt, sum))
```

با اجرای کُدهای بالا نتیجه حاصل می‌شود.

```
placebo drug drug+
12 18 13
```

۱-۹-۳ توابع

تابع یک بخش از برنامه است که وظیفه خاصی را انجام می‌دهد. دستور عمومی نوشتن یک تابع به شرح زیر است.

```
functionName <- function(arg1, arg2, ···) {
          do this}
```

مثال زیر تابعی است که قضیه فیثاغورث را نشان می‌دهد.

```
> hypot <- function(a, b) sqrt(a^2 + b^2)
> hypot(3,4)
[1] 5
```

در برنامه فوق موارد زیر درخور توجه است.

- متغیرهای موقتی a, b با اعداد 3, 4 ساخته می‌شود.
- با محاسبه sqrt(a^2 + b^2) عدد 5 بدست می‌آید.
- وقتی محاسبه تمام شد، تعریف موقت a, b حذف می‌شود.

مثال: تابع زیر را در نظر بگیرید که میانگین و انحراف معیار داده‌ها را محاسبه می‌کند.

```
> mystats <- function(x)
+ {
+ mymean <- mean(x)
+ mysd <- sd(x)
+ c(mean=mymean,sd=mysd)
```

```
+ }
```

اگر داده‌های زیر به تابع فوق داده شود مقادیر زیر نتیجه می‌گردد.

```
> my_data <- c(1,2,3,4,5)
> mystats(my_data)


 mean       sd
3.000000 1.581139
```

همین تابع با استفاده از تابع list() در زبان R به‌کار گرفته می‌شود.

```
> mystats <- function (x)
+ {
+ myinput <- x
+ mymean <- mean(x)
+ mysd <- sd(x)
+ list(data = myinput, mean = mymean, sd = mysd)
+ }
```

اگر داده‌های زیر به تابع فوق داده شود مقادیر زیر نتیجه می‌گردد.

```
> my_data <- c(1,2,3,4,5)
> mystats(my_data)


$data
[1] 1 2 3 4 5

$mean
[1] 3

$sd
[1] 1.581139
```

در زبان R می‌توان برنامه‌های بازگشتی نیز داشت. به تابع فاکتوریل توجه کنید که به‌صورت بازگشتی نوشته شده است.

```
> Fact <- function(n) if (n == 1) 1 else n * Fact(n - 1)
> Fact(5)
[1] 120
```

۱-۹-۳-۱ آرگومان اجباری و اختیاری

وقتی یک تابع در زبان R فراخوانده می‌شود، تعریف تابع معین می‌کند که کدام آرگومان اجباری و کدام یک اختیاری است. در مثال زیر آرگومان x اجباری است و اگر نباشد تابع خطا می‌دهد و آرگومان k اختیاری است و مقدار پیش‌فرض 2 را دارد.

```
> power <- function(x, k=2) {
+ x^k
+ }
> power(5)
[1] 25
> power()
Error in power() : element 1 is empty;
    the part of the args list of '^' being evaluated was:
    (x, k)
```

اما می‌توان برای آرگومان k مقدار متفاوتی را معین نمود.

```
> power(5,3)
[1] 125
```

۱-۹-۳-۲ آرگومان «...»

آرگومان سه نقطه می‌تواند برای انتقال آرگومان‌های یک تابع به تابع دیگر استفاده شود. فرض کنید می‌خواهید تابع کوچکی بنویسید که تابع سینوس را از 0 تا xup رسم کند.

```
> plotsin <- function(xup=2*pi,...)
+ {
+ x <- seq(0, xup, l=100)
+ plot(x, sin(x), type="l",...)
+ }
> plotsin(col="red")
```

تابع plotsin هر آرگومانی را که می‌توان در تابع plot (نظیر col، lwd، xlab و سایر) را می‌پذیرد بدون آن که مجبور باشید که آن را در تابع plotsin مشخص کنید.

۱-۹-۳-۳ نام آرگومان‌ها

اصولاً آرگومان‌های یک تابع می‌توانند دارای نام و یا کلید واژه[18] باشند. برای درک مناسب‌تر این مطلب به مثال زیر توجه کنید.

```
> sumsq <- function(a, xv=x, yv=y)
+ { yf <- exp(-a*xv)
+ sum((yv-yf)^2) }
```

18. keyword

همان‌طور که در تابع فوق مشاهده می‌کنید آرگومان‌های گوم و سوم تابع به ترتیب دارای نام‌های x و y هستند اما آرگومان a نام ندارد، یعنی مانند xv و yv است. البته ذکر این نکته ضروری است که اگر این دو آرگومان نیز مانند آرگومان a بودند تابع بالا کار می‌کرد و مشکلی از این حیث وجود نداشت. اما این امکان اضافی دارای کاربردی است که در زیر به آن پرداخته می‌شود. اکنون مقدار تابع sumsq(1,2,3) محاسبه می‌شود.

```
> sumsq <- function(a,xv=x,yv=y)
+ { yf <- exp(-a*xv)
+ sum((yv-yf)^2) }
> sumsq(1,2,3)
[1] 8.206304
```

اکنون مقدار تابع sumsq(1,3,2) محاسبه می‌شود یعنی جای مقادیر x و y عوض می‌شود و انتظار می‌رود که نتیجه تابع نسبت به دفعه قبل فرق کند، چون جای مقادیر x و y عوض شده است.

```
> sumsq <- function(a,xv=x,yv=y)
+ { yf <- exp(-a*xv)
+ sum((yv-yf)^2) }
> sumsq(1,3,2)
[1] 3.80333
```

اما اکنون از نام آرگومان‌های دوم و سوم استفاده می‌شود و تعویض محل قرارگرفتن x و y نیز صورت می‌گیرد.

```
> sumsq <- function(a,xv=x,yv=y)
+ { yf <- exp(-a*xv)
+ sum((yv-yf)^2) }
> sumsq(1, y=3, x=2)
[1] 8.206304
```

ملاحظه می‌گردد که به‌رغم جابجای محل قرارگرفتن آرگومان‌های x و y جواب مانند sumsq(1,2,3) است. بنابراین می‌توان نتیجه گرفت که با به‌کار بردن نام آرگومان‌ها تقدم و تأخر آنها در نتیجه تابع مؤثر نیست. اما بدون ذکر نام آرگومان‌ها ترتیب قرارگرفتن آنها حائز اهمیت است و در صورت عدم رعایت آن نتیجه متفاوت خواهد بود. تمام توابع کتابخانه‌ای و یا پیش‌ساخته در زبان R با همین قاعده عمل می‌کنند.

۱-۹-۳-۴ متغیرهای محلی

در داخل تابع اگر به متغیری مقداری را منسوب نمایید آن متغیر محلی است، مگر این که به صراحت آن را جهانی «<<-» تعریف کنید. به عبارت دیگر متغیر محلی، خارج از تابع مقدارش روی متغیر هم‌نامش جایگزین نمی‌شود. ضمناً اگر تابع خاتمه یابد مقدار آن از بین می‌رود مگر آن که آخرین دستور تابع یک انتساب باشد.

در مثال زیر، ابتدا متغیر x مقدار صفر می‌گیرد. در داخل تابع مقدار x برابر 3 می‌گردد. اجرای تابع تأثیری در متغیر جهانی (بیرونی) نمی‌گذارد.

```
> x <- 0
> functionx <- function() {
```

```
+ x <- 3
+ }
> functionx()
> x
[1] 0
```

اکنون مثال بالا به صورت جهانی تعریف می‌شود.

```
> x <- 0
> functionx <- function() {
+ x <<- 3
+ }
> functionx()
> x
[1] 3
```

آرگومان تابع می‌تواند هر شی باشد، حتی یک تابع، به مثال زیر توجه کنید.

```
> test <- function(n, fun)
+ {
+ u <- runif(n)
+ fun(u)
+ }
> test(3,sin)
[1] 0.7537332 0.8033265 0.3290288
```

۱-۹-۳-۵ ساخت یک عملگر دوتایی تعریف شده کاربر

در زبان R می‌توان عملگر دوتایی مورد نظر را ساخت. برای این کار از علامت «%» مانند %anything% استفاده می‌شود. ضمناً در تعریف عملگر، باید نام آن در داخل کوتیشن قرار گیرد. برای مثال:

```
"%anything%" <- function(x,y) {···}
```

قبل از شروع مثال زیر، به عملکرد توابع union() و setdiff() توجه کنید.

```
> x <- c(1,2,5)
> y <- c(5,1,8,9)
> union(x,y)
[1] 1 2 5 8 9
> setdiff(x,y)
[1] 2
> setdiff(y,x)
[1] 8 9
```

اکنون عملگری ساخته می‌شود که اختلاف دو مجموعه را به صورت متقارن محاسبه می‌کند.

```
> "%sdf%" <- function(a,b) {
+ sdfxy <- setdiff(x,y)
+ sdfyx <- setdiff(y,x)
```

```
+ union(sdfxy,sdfyx)
+ }
> x %sdf% y
[1] 2 8 9
```

مثال: برای محاسبات معمولاً دو رویکرد مرسوم است. روش تکرار[19] و روش بازگشتی[20]، مثال زیر نحوه کار هر دو را نشان می‌دهد. فرمول زیر محاسبه عدد احتمال نرمال را برای $z>0$ نشان می‌دهد.

$$\Phi(z) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n z^{2n+1}}{n! 2^n (2n+1)}$$

اول روش تکرار دنبال می‌شود که کدهای آن به شرح زیر است.

```
> Phi1 <- function(z) {
+ sum = z
+ nfac = 1
+ for (n in 1:30) {
+ nfac = n * nfac
+ sum = sum + (-1)^n * z ^ (2 * n + 1) / + (nfac * 2^n * (2 * n + 1))}
+ 0.5 + sum / sqrt(2 * pi)
+ }
> Phi1(1.96)
[1] 0.9750021
```

مساله بالا به روش بازگشتی در زیر ملاحظه می‌شود.

```
> Phi2 <- function(z)
+ {
+ n2p1 = 1; mult = -z^2/2
+ term = z; sum = term
+ for (n in 1:30) {
+ n2p1 = n2p1 + 2
+ term = term * mult / n
+ sum = sum + term / n2p1
+ }
+ 0.5 + sum / sqrt(2 * pi)
+ }
> Phi2(1.96)
[1] 0.9750021
```

۱-۹-۳-۶ ذخیره تابع نوشته شده و فراخوان آن

پس از این که تابع تعریف شده کاربر نوشته شد و کامل گردید و از درستی آن اطمینان حاصل شد. آنگاه می‌توان آن به‌عنوان یک R فایل با نامی دلخواه در محل مورد نظر برای استفاده‌های بعدی ذخیره نمود. اکنون برای استفاده از آنها ابتدا باید دستور source("نام و مسیر R فایل") را اجرا کرد و سپس تابع مورد نظر را فراخواند. در اینجا دو نکته قابل توجه است.

19. iteration 20. recursion

- اول این که نام R فایل و نام تابع یکسان نباشد چون ممکن است که در یک R فایل چندین تابع با نام‌های گوناگون ذخیره شود.
- اگر برای هر تابع یک R فایل مجزا ایجاد شد. به ازاء هر یک از R فایل‌ها باید از دستور source استفاده شود و سپس توابع متفاوت را فراخواند.

اکنون به مثال زیر توجه کنید. تابعی برای محاسبه میانگین و انحراف معیار نوشته شده است که مثلاً در فایلی به نام mean_sd ذخیره گردیده است.

```
mean.sd <- function (x, na.rm = TRUE)
{
if (na.rm) x <- x[!is.na(x)]
n <- length(x)
if (n < 2) {
cat("small number of data", "\n")
return(c(Mean = NA, SD = NA))
}
xbar <- sum(x)/n
sd <- sqrt(sum((x - xbar)^2)/(n - 1))
c(Mean = xbar, SD = sd)
}
```

حالا باید تابع source() به‌صور زیر اجرا گردد.

```
> source("D:\\R_files\\mean_sd.R")
```

سپس داده‌های مورد نظر برای محاسبه میانگین و انحراف معیار به تابع داده می‌شود.

```
> x <- c(1,2,3)
> mean.sd(x)
```

حاصل محاسبه به‌صورت زیر است.

```
Mean    SD
 2      1
```

۱-۹-۴ برنامه‌نویسی بازگشتی[21]

این نوع برنامه‌نویسی، تکنیک توانایی در توابع می‌باشد. برنامه بازگشتی می‌تواند خودش را صدا بزند و برای الگوریتم‌هایی که خصلت بازگشتی دارند، مفید است.

مثال: فاکتوریل یک می‌تواند یکی از مصادیق روابط بازگشتی باشد، زیرا $n! = n \times ((n-1)!)$ است. توجه داشته باشید که با استفاده از دستور cat می‌توان پاره‌ای از بازخوردهای تابع را به‌دست داد. اکنون به کُدهای زیر توجه کنید.

21. Recursive programming

```
> nfact <- function(n) {
+ # calculate n factorial
+ if (n == 1) {
+ cat("called nfact(1)\n")
+ return(1)
+ } else {
+ cat("called nfact(", n, ")\n", sep = "")
+ return(n*nfact(n-1))
+ }
+ }
> nfact(6)
```

با اجرای برنامه بالا نتیجه زیر حاصل می‌گردد.

```
called nfact(6)
called nfact(5)
called nfact(4)
called nfact(3)
called nfact(2)
called nfact(1)
[1] 720
```

مثال: در اینجا قرار است که اعداد اول در یک بازه مشخص شناسایی گردد. این کار با استفاده از روش برنامه‌نویسی بازگشتی صورت می‌گیرد. الگوریتم کار به‌صورت زیر است.

۱) با لیست $2, 3, \cdots, n$ شروع کنید. بزرگترین عدد معلوم اول p=2 است.

۲) تمام عناصری که مضربی از p هستند به جزء خودش حذف می‌گردد.

۳) مقدار p را به کوچکترین عدد باقی‌مانده در لیست که از p جاری بزرگتر است افزایش دهید.

۴) اگر p بزرگتر از $\sqrt{n}$ بود، برنامه خاتمه می‌یابد وگرنه برنامه به گام دوم باز می‌گردد.

اکنون به کُدهای برنامه توجه کنید.

```
> primesieve <- function(sieved, unsieved) {
+ # finds primes using the Sieve of Eratosthenes
+ # sieved: sorted vector of sieved numbers
+ # unsieved: sorted vector of unsieved numbers
+ # cat("sieved", sieved, "\n")
+ # cat("unsieved", unsieved, "\n")
+ p <- unsieved[1]
+ n <- unsieved[length(unsieved)]
+ if (p^2 > n) {
+ return(c(sieved, unsieved))
+ } else {
+ unsieved <- unsieved[unsieved + sieved <- c(sieved, p)
+ return(primesieve(sieved, unsieved))
```

```
+ }
+ }
> primesieve(c(), 2:200)
```

با اجرای برنامه بالا نتیجه زیر حاصل می‌گردد.

```
 [1]   2   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67
[20]  71  73  79  83  89  97 101 103 107 109 113 127 131 137 139 149 151 157 163
[39] 167 173 179 181 191 193 197 199
```

۱-۱۰ محاسبه مشتق، انتگرال و معادلات دیفرانسیل

در زبان R می‌توان عملیات ریاضی مانند مشتق، انتگرال و معادلات دیفرانسیل را انجام داد. همان‌طورکه در دنباله خواهد آمد محاسبه پاره‌ای از آنها (مشتق و انتگرال) در هسته R انجام می‌شود و برای معادلات دیفرانسیل نیاز به بارگذاری یک بسته نرم‌افزاری است.

۱-۱۰-۱ مشتق

در R می‌توان مشتق توابع ساده را به‌صورت تحلیلی به‌دست آورد. برای این کار از تابعی به نام D() استفاده می‌شود که آرگومان اول آن عبارتی است که قرار است از مشتق گرفته شود و آرگومان دوم متغیری است که مشتق‌گیری نسبت به آن انجام می‌گردد. اکنون به مثال‌های زیر وتجه کنید.

```
> D(expression(2*x^3), "x")
2 * (3 * x^2)

> D(expression(log(x)), "x")
1/x

> D(expression(a*exp(-b * x)), "x")
-(a * (exp(-b * x) * b))

> D(expression(a/(1+b*exp(-c * x))), "x")
a * (b * (exp(-c * x) * c))/(1 + b * exp(-c * x))^2

> trig.exp <- expression(sin(cos(x + y^2)))
> D(trig.exp, "x")
-(cos(cos(x + y^2)) * sin(x + y^2))
```

۱-۱۰-۲ انتگرال

برای محاسبه انتگرال به‌صورت عددی از تابع integrate() استفاده می‌شود. در این تابع وجود سه زبان R ضروری است. اول تابعی است که قرار است از آن انتگرال گرفته شود. دوم حد پایین انتگرال و سومی حد بالای انتگرال است. اکنون به مثال‌های زیر توجه کنید.

```
> f <- function(x) {x^2+1}
> integrate(f, lower=0, upper=1)
1.333333 with absolute error < 1.5e-14
```

```
> g <- function(x) {1/(1+x^2)}
> integrate(g, -Inf, Inf)
3.141593 with absolute error < 5.2e-10
```

در اینجا ذکر چند نکته ضروری است. اول این که نام تابع در اختیار کاربر است. دوم این که گذاشتن نام‌های lower و upper اختیاری است. سوم این که Inf به معنی بی‌نهایت است و حتماً حرف اول آن یعنی I باید بزرگ نوشته شود وگرنه خطا ایجاد می‌شود.

```
> h <- function(x) {1/(pi*(1+x^2))}
> integrate(h, -Inf, Inf)
1 with absolute error < 1.6e-10
```

۱-۱۰-۲-۱ انتگرال چندگانه

برای محاسبه انتگرال چندگانه عددی باید از تابع integrate() به‌صورت مکرر استفاده شود. برای مثال به انتگرال دوگانه زیر توجه کنید.

$$\int_0^3 \int_1^2 x^2 y \, dy \, dx$$

برای محاسبه عددی انتگرال بالا به کُدهای زیر توجه کنید.

```
> # Iterated Integral
> integrate(function(x) {
+ sapply(x, function(x) {
+ integrate(function(y) x^2*y, 1, 2)$value
+ })
+ }, 0, 3)
13.5 with absolute error < 1.5e-13
```

۱-۱۰-۳ معادله دیفرانسیل

در علوم و مهندسی با مسائلی مواجه می‌شویم که لازم است در آنها به حل معادلات دیفرانسیل پرداخته شود. در زبان برنامه‌نویسی R یک بسته نرم‌افزاری با نام deSolve موجود می‌باشد که دارای این قابلیت است که می‌تواند

به حل انواع معادلات دیفرانسیل بپردازد و به‌کمک آن می‌توان به‌راحتی یک مساله را حل نمود و دیگر نیازی به استفاده از الگوریتم‌های حل معادلات که گاهی کار با آنها بسیار سخت و طولانی می‌شود، نخواهد بود. در این قسمت ابتدا به معرفی یک مساله مرتبط با مهندسی آب که در آن نیاز به حل یک معادله دیفرانسیل معمولی می‌باشد، پرداخت می‌شود و سپس با ارائه مثالی چگونگی حل آن با استفاده از زبان R تشریح می‌گردد.

یک سرریز کناری عبارتست از یک سرریز با جریان آزاد که در کناره کانال و به موازات آن تعبیه شده و اجازه می‌دهد تا در موقعی که ارتفاع آب بالاتر از تاج سرریز است، مقداری از آب از روی آن خارج شود. این نوع سرریزها در انحراف آب اضافی در سیستم‌های جمع‌آوری آب شهری، در کنترل انحراف سیلاب‌ها و . . . مورد استفاده فراوانی دارند.

De Marchi در سال 1934 برای به‌دست آوردن معادله پروفیل جریان آب برروی سرریزهای کناری فرضیات زیر را در نظر گرفت:

۱) کانال مستطیلی و منشوری باشد.

۲) سرریز کناری دارای طول کوتاه بوده و انرژی مخصوص بین دو مقطع ثابت است.

۳) سرریز کناری معادل یک سرریز لبه تیز بوده که هوادهی کامل صورت گرفته و آب به‌صورت آزاد خارج می‌شود.

۴) ضریب تصحیح انرژی مساوی با یک است.

سرریز کناری می‌تواند برروی یک کانال با شیب بسیار کم، ملایم و تند تعبیه گردد و معادله‌ای که De Marchi در نهایت بدان دست یافته برای هریک از سه کانال ذکر شده اعتبار دارد و به‌صورت یک معادله دیفرانسیل با رابطه زیر بیان می‌شود:

$$\frac{dy}{dx} = \frac{4}{3}\frac{C_M}{B}\sqrt{\frac{(E-y)(y-W)^3}{3y-2E}}$$

که در رابطه بالا

B و W: عرض کانال و ارتفاع سرریز بوده و دارای مقدار مشخصی می‌باشند.

E: انرژی مخصوص بوده و با توجه به اینکه ارتفاع جریان در ابتدای سرریز مشخص می‌باشد، مقدار آن نیز معلوم است.

C_M: ضریب شدت جریان نام دارد و به‌صورت رابطه‌ای برحسب عدد فرود بیان می‌شود و مقدار آن نیز مشخص می‌باشد.

y: ارتفاع آب برروی سرریز می‌باشد که مجهول بوده و با حل معادله دیفرانسیل فوق تعیین شده و به این طریق پروفیل آب برروی سرریز کناری به‌دست می‌آید.

به‌عنوان مثال، پارامترهای زیر که مربوط به یک سرریز کناری با طول 2.534، تعبیه شده برروی یک کانال مستطیلی با شیب بسیار کم می‌باشند را در نظر بگیرید.

$$Cm = 0.502 \quad B = 2 \quad E = 0.4933 \quad W = 0.33$$

برای حل معادله دیفرانسیل در R به‌صورت گام‌های زیر عمل می‌شود:

```
parms <- c(Cm, B, E, W)
```

در مرحله بعد لازمست یک تابع (function) تعریف شود که در آن معادله دیفرانسیل معرفی گردد؛ به‌صورت زیر:

```
func <- function(x, y, parms) {
dy <- (4/3)*(Cm/B)*(((E-y)*(y-W)^3)/(3*y-2*E))^(1/2)
list(c(dy))
}
```

در تعریف رابطه فوق، سه آرگومان به‌کار گرفته شده است:

x، طول فواصلی است که به‌ازای آن مقدار y که همان ارتفاع آب روی سرریز است، مشخص می‌گردد؛ یعنی به‌ازای فاصله x از ابتدای سرریز، مقدار y مشخص می‌شود. در روش‌های عددی حل معادلات دیفرانسیل، مقدار x همان مش‌بندی‌هایی است که انجام می‌گیرد. برای تعریف x می‌توان از دستور زیر استفاده نمود:

```
x <- seq(0,2.534, 0.1)
```

y، متغیر حالت نامیده شده و مجهول مساله است و با حل معادله دیفرانسیل، مقدار آن مشخص می‌شود.

parms، همان پارامترهای معادله دیفرانسیل می‌باشند.

همانگونه که مشاهده می‌شود، نتیجه تابع تعریف شده در بالا به‌صورت list برگردانده شده که این به‌خاطر پیش‌فرض حل معادلات دیفرانسیل در Package می‌باشد. حال برای حل معادله مورد نظر که یک معادله دیفرانسیل معمولی می‌باشد، پس از فراخوانی بسته نرم‌افزاری با نام deSolve با استفاده از تابع ()ode می‌توان معادله را حل نموده و در قالب data frame در متغیری با نام out ذخیره نمود.

```
> library(deSolve)
> Cm <- 0.502
> B <- 2
> E <- 0.4933
> W <- 0.33
> y <- c(y=0.44)
> x <- seq(0, 2.534, 0.1)
> parms <- c(Cm, B, E, W)
> func <- function(x, y, parms) {
+ dy <- (4/3)*(Cm/B)*(((E-y)*(y-W)^3)/(3*y-2*E))^(1/2)
+ list(c(dy))
}
> out <- as.data.frame(ode(y, times=x, func, parms))
> head(out)
```

حاصل اجرای کُدهای بالا به‌صورت زیر است.

```
 time          y
```

```
1 0.0 0.4400000
2 0.1 0.4404881
3 0.2 0.4409762
4 0.3 0.4414641
5 0.4 0.4419519
6 0.5 0.4424395
```

۱۱-۱ بسته‌های نرم‌افزاری[۲۲]

یکی از ویژگی‌های بسیار مهم زبان R داشتن بسته‌های نرم‌افزاری[۲۳] فراوانی است که به هسته[۲۴] و یا پایه R محلق می‌شوند. برای توجه به این مؤلفه ممتاز باید گفت که بیش از 2000 بسته نرم‌افزاری وجود دارد که عددی درخور توجه است. نکته‌ای که به این مزیت می‌افزاید این است که تمام بسته‌های موصوف به‌صورت رایگان است و دارای متن باز[۲۵] هستند. ضمناً به‌روز نیز می‌شوند.

اگر در خلال این فصل توجه کرده باشید به تفاریق به بسته‌هایی اشاره شده است. اما توضیح کافی در خصوص به‌دست آوردن، نصب، ارتباط و سایر جزئیات آنها درج نشده است، که اکنون به این مهم پرداخته می‌شود.

وقتی نرم‌افزارR روی کامپیوتر شما نصب می‌گردد، در واقع تعدادی بسته‌نرم‌افزاری نیز خود به‌خود نصب می‌شوند. بحث بسته‌های نرم‌افزاری آنقدر اهمیت دارد که در R در قسمت نوار ابزار بالای صفحه، منوی Packages را مشاهده می‌کنید. اگر روی این منو کلیک شود، فهرستی از گزینه‌ها به‌صورت زیر ملاحظه می‌شود.

```
Local package ···
─────────────────────────────────
Set CRAN mirror
Select repositories ···
Install package(s) ···
Update package(s)
─────────────────────────────────
Install package(s) from local zip files ···
```

اگر گزینه Local package ··· را بزنید، فهرست بسته‌های بارگذاری شده در R را به شما بر حسب ترتیب حروف الفباء نشان می‌دهد. ضمنا می‌توان از تابع زیر نیز استفاده نمود که علاوه بر فهرستشان مسیر نصب آنها را نیز به‌دست می‌دهد.

```
> library()
```

اگر به فهرست گفته به دقت نگاه کنید، ملاحظه می‌شود که یک نام base وجود دارد که توضیح آن The R Base Package می‌باشد. در این صورت به تمام توابع و داده‌های آن دسترسی وجود دارد. اما بقیه بسته‌ها هر

22. Packages

۲۳. منظور از بسته‌های نرم‌افزاری، کتابخانه‌هایی هستند که شامل توابع و داده‌ها می‌باشند.

24. core 25. open source

کدام دارای نام مجزا و مشخص می‌باشند و تفاوت آنها با بسته‌های پایه در این است که به‌رغم بارگذاری دسترسی به توابع و داده‌های آنها میسر نیست. برای دسترسی به موارد اخیر از تابع (نام بسته)library باید استفاده شود. البته از تابع ()require بجای تابع ()library نیز می‌توان استفاده نمود. به مثال زیر توجه کنید.

```
> library(MASS)
```

از این به بعد دسترسی به توابع و داده‌های بسته MASS میسر است. اگر از R خارج شوید ارتباط قطع می‌گردد و دوباره باید با استفاده از تابع ()library ارتباط گفته شده برقرار گردد.

واضح است که تمام بسته‌های R به‌صورت پیش‌فرض در آن بارگذاری نمی‌شوند و کاربر حسب نیاز بسته‌های خود را انتخاب نموده و بارگذاری می‌کند و در صورت کاربست ارتباط لازم را نیز برقرار می‌سازد. اما تفصیل این مراحل چگونه صورت می‌گیرد.

۱) ابتدا در یک موتور جستجو مانند google یا شبیه آن، نام بسته مورد نظر و کلمه package را تایپ کنید. به‌عنوان مثال e1071 package در نظر گیرید، پس از جستجو عبارت CRAN - Package e1071[۲۶] (شبکه بایگانی فراگیر R) را ملاحظه می‌کنید. روی آن کلیک کنید. صفحه‌ای مختص این بسته گشوده می‌شود. در قسمت :Downloads این صفحه، در مقابل گزینه :Windows binary فایل فشرده e1071.zip را download کنید. اکنون فایل فشرده روی کامپیوتر شما قرار دارد. البته در این صفحه یک فایل pdf جهت راهنمایی برای بسته مورد نظر وجود دارد که می‌توان آن را نیز download نمود.

۲) توجه داشته باشید که به هیچ‌وجه سعی نکنید که فایل فشرده گفته شده را باز کنید. برای باز نمودن فایل گفته شده از خود R کمک بگیرید. بدین معنی که در منوی Packages گزینه

Install package(s) from local zip files ···

را انتخاب نموده و روی آن کلیک کنید. با این کار پنجره‌ای به نام Select files گشوده خواهد شد و شما می‌توانید به محل ذخیره بسته مورد نظر در کامپیوتر خود بروید و آن را انتخاب نموده و کلید Open را بزنید. آنگاه در R پیغام زیر ظاهر می‌گردد.

```
package 'e1071' successfully unpacked and MD5 sums checked
```

به این ترتیب بسته e1071 بارگذاری می‌شود. یعنی شما نام این بسته را در فهرست بسته‌های R خود مشاهده می‌کنید.

برای بارگذاری یک بسته نرم‌افزاری راه حل دیگری نیز وجود دارد که بجای استفاده از منوها و پنجره‌های گوناگون می‌توان از تابع ("مسیر و نام بسته")install.packages استفاده نمود. به مثال زیر توجه کنید.

```
> install.packages("D:\\R_files\\packages\\ e1071_1.6.zip")
inferring 'repos = NULL' from the file name
package 'e1071' successfully unpacked and MD5 sums checked
```

توجه داشته باشید که حتماً پسوند بسته یعنی zip و شماره نسخه بسته یعنی 1.6 باید قید شود. به عبارت دیگر نام بسته مورد نظر باید به‌طور کامل نوشته شود وگرنه با خطا مواجه می‌شوید.

26. **C**omprehensive **R** **A**rchive **N**etwork (**CRAN**)

۳) اما هنوز به توابع و داده‌های این بسته دسترسی ندارید و باید از دستور زیر استفاده کنید.

```
> library(e1071)
```

برای مشاهده توابع بسته مورد نظر می‌توان از دستور زیر استفاده نمود.

```
> library(help=e1071)
```

۴) برای استفاده از بعضی بسته‌ها نیاز به بارگذاری یک یا چند بسته‌ی دیگر است، لذا قبل از استفاده از بسته مورد نظر، باید این امر را انجام داد. در غیر این صورت، R نیاز مذکور را به‌صورت اخطار نشان می‌دهد. به مثال زیر توجه کنید.

```
> library(e1071)
Loading required package: class
```

همان‌طورکه در پیام بالا ملاحظه می‌گردد برای استفاده از بسته e1071 به بسته‌ای به نام class نیاز است. که در زیر اضافه می‌گردد.

```
> library(class)
> library(e1071)
>
```

مشاهده می‌شود که دیگر پیغامی ظاهر نمی‌گردد.
اکنون اگر بخواهید کُدهای یک تابع را از بسته مورد نظر مشاهده کنید کافی است که نام تابع را تایپ نموده و کلید Enter را بزنید.

۵) اگر احیاناً به بسته‌ای نیاز ندارید و می‌خواهید آن را از فهرست گفته شده حذف کنید، کافی است از دستور زیر استفاده کنید. توجه داشته باشید که در اینجا ذکر شماره نسخه و یا پسوند برای بسته مورد نظر لازم نیست و تنها می‌توان به نام بسته بسنده نمود.

```
> remove.packages("e1071")
```

۶) برای مشاهده داده‌های یک بسته به‌صورت اختصاصی می‌توان از دستور زیر استفاده نمود.

```
> data(package="package name")
```

اکنون به مثال زیر توجه کنید.

```
> data(package="MASS")
```

فصل دوم

رسم نمودار

در زبان R می‌توان نمودارهای متفاوتی را رسم نمود. برای این‌که بتوانید بخشی از آن‌ها را ملاحظه کنید، دستورات demo(graphics) و یا demo(persp) را در R تایپ کنید. در زبان R دوگونه تابع برای رسم نمودارها موجود است.

- توابع سطح بالا[1] که می‌توانند نمودار جدیدی را ایجاد نمایند.
- توابع سطح پایین[2] که می‌توانند عناصری را به نمودار موجود بیافزایند.

برای آشنایی با دستورات رسم نمودار در زبان R، تعدادی از آن‌ها ارائه خواهد شد.

۲-۱ توابع نموداری

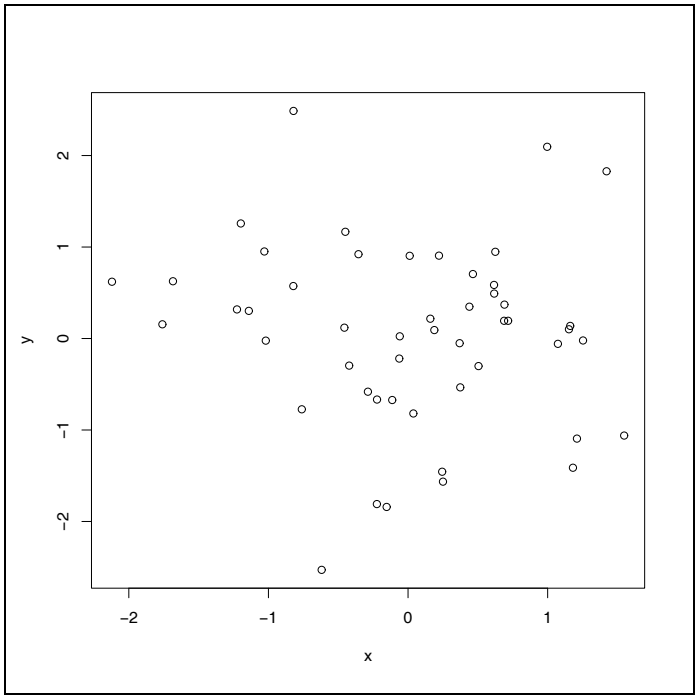
در این قسمت تعدادی از توابع سطح بالای نموداری ملاحظه می‌گردد. تابعی که کاربرد فراوانی در ترسیم دارد، تابع plot() است. در مثال زیر با استفاده از دو داده‌ی برداری می‌توان یک نمودار پراکَنش رسم نمود.

```
> x <- rnorm(50)
```

1. high-level plotting functions 2. low-level plotting functions

```
> y <- rnorm(50)
> plot(x,y)
```

پس از اجرای دستورات فوق شکل زیر حاصل می‌گردد. که اگر دستور سطح پایین زیر را اضافه کنید، می‌توان



شکل ۲-۱: نمایش یک نمودار پراکَنش

یک عنوان برای شکل داشت.

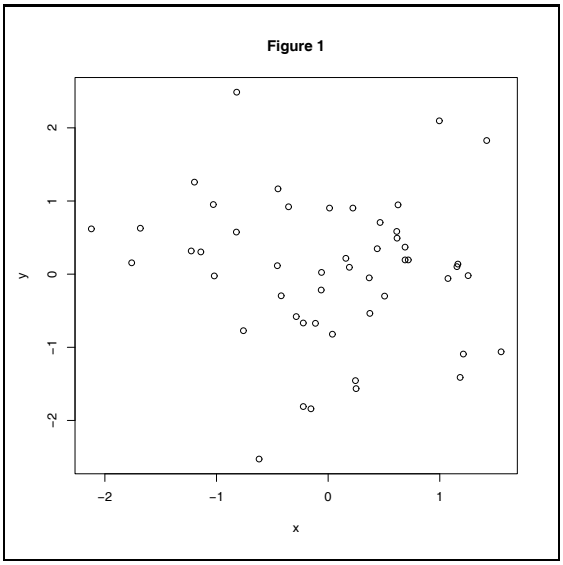
```
> title("Figure 1")
```

همان‌طور که ملاحظه می‌کنید، شکل ترسیم شده به‌صورت پیش‌فرض نقطه‌ای است. اما می‌توان انواع دیگر شکل‌ها را، با استفاده از خاصیت type در تابع plot() داشت. اکنون برای سایر موارد به کُدها و ترسیم‌های زیر توجه کنید.

```
> x <- seq(-5,5,1)
> y <- x^2
> par(mfrow=c(1,2))
> plot(x,y); text(-0.5,20,"without type")
> plot(x,y, type='b'); text(-0.5,20,"type='b'")
```

شکل ۲-۳ نتیجه اجرای کُدهای بالا است.

```
> x <- seq(-5,5,1)
> y <- x^2
> par(mfrow=c(1,2))
> plot(x,y, type='c'); text(-0.5,20,"type='c'")
```



شکل ۲-۲: نمایش یک نمودار پراکَنش با عنوان

```
> plot(x,y, type='h'); text(-0.5,20,"type='h'")
```

شکل ۲-۴ نتیجه اجرای کُدهای بالا است.

```
> x <- seq(-5,5,1)
> y <- x^2
> par(mfrow=c(1,2))
> plot(x,y, type='l'); text(-0.5,20,"type='l'")
> plot(x,y, type='o'); text(-0.5,20,"type='o'")
```

شکل ۲-۵ نتیجه اجرای کُدهای بالا است.

```
> x <- seq(-5,5,1)
> y <- x^2
> par(mfrow=c(1,2))
> plot(x,y, type='s'); text(-0.5,20,"type='s'")
> plot(x,y, type='S'); text(-0.5,20,"type='S'")
```

شکل ۲-۶ نتیجه اجرای کُدهای بالا است.

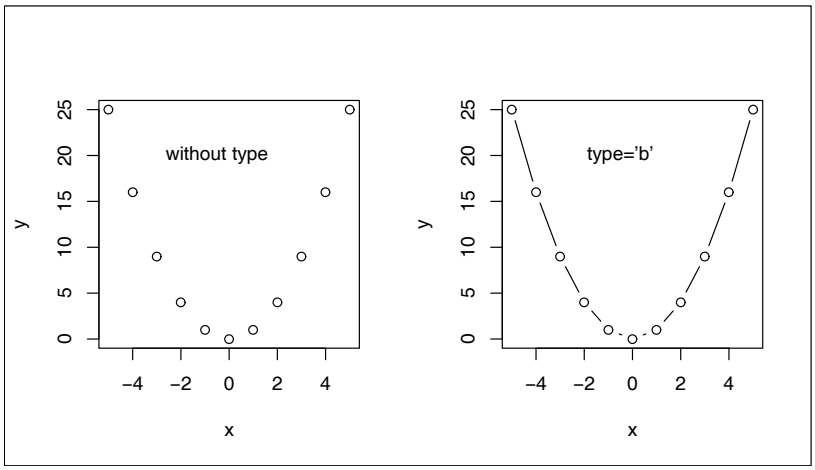
۲-۱-۱ تابع curve()

این تابع زبان R می‌تواند توابع پیوسته را روی یک فاصله معین رسم نماید. شکل کلی آن به صورت زیر است.

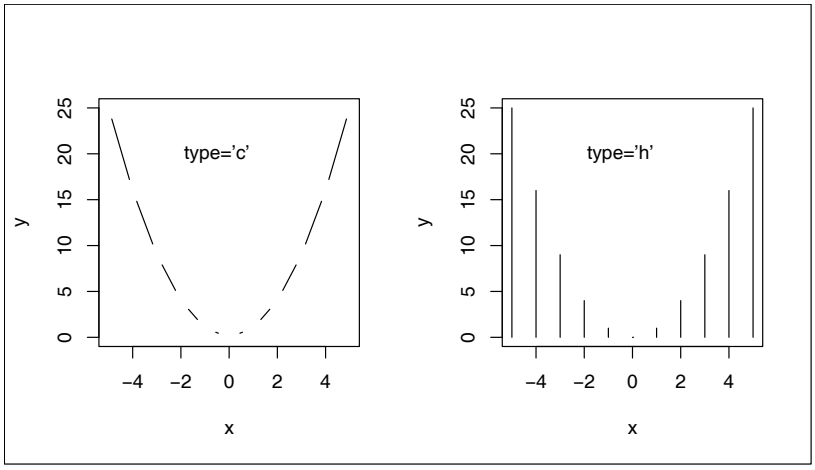
curve(expr, from, to, add = FALSE, ...)

شرح آرگومان‌ها:

expr: عبارتی که بر حسب x نوشته شده است.



شکل ۲-۳: نمایش نمودار با type='b'



شکل ۲-۴: نمایش نمودار با type='c' و type='h'

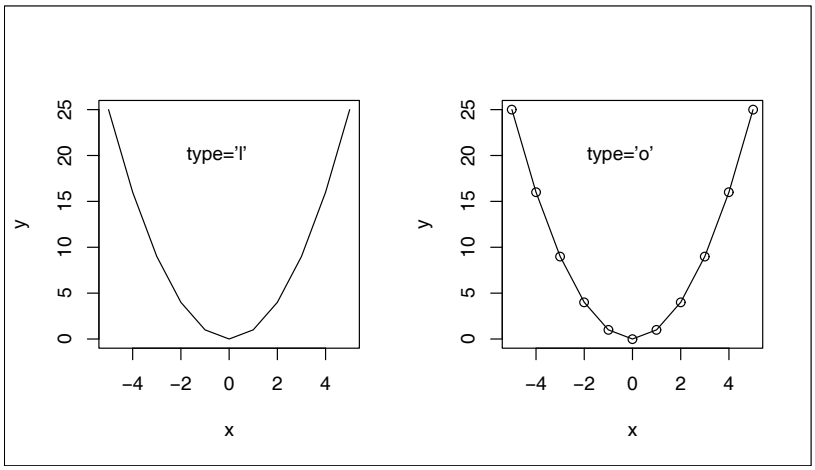
from, to: دامنه‌ای است که تابع باید روی آن رسم گردد.
add: منطقی است و اگر «TRUE» باشد شکل تابع به شکل حاضر اضافه می‌شود.
مثال: نمودار تابع سینوس را از 0 تا 2π رسم کنید.

```
> curve(sin(x), from=0, to=2*pi)
```

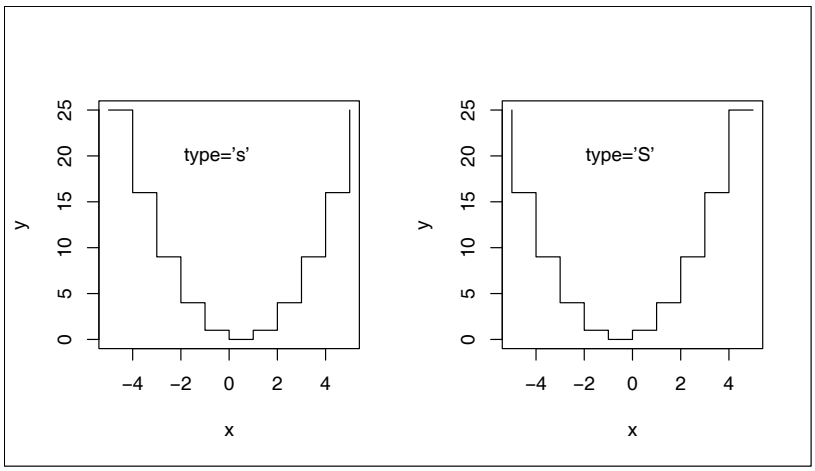
شکل ۲-۷ آن را نشان می‌دهد.
مثال: نمودار تابع نرمال استاندارد را از 3- تا 3 رسم کنید.

```
> curve(dnorm(x), from=-3, to=3)
```

شکل ۲-۸ آن را نشان می‌دهد.
مثال: در اینجا تابع احتمال نرمال که توسط تابع curve() تهیه شده است به هیستوگرام آن اضافه می‌گردد. به



شکل ۲-۵: نمایش نمودار با type='l' و type='o'



شکل ۲-۶: نمایش نمودار با type='s' و type='S'

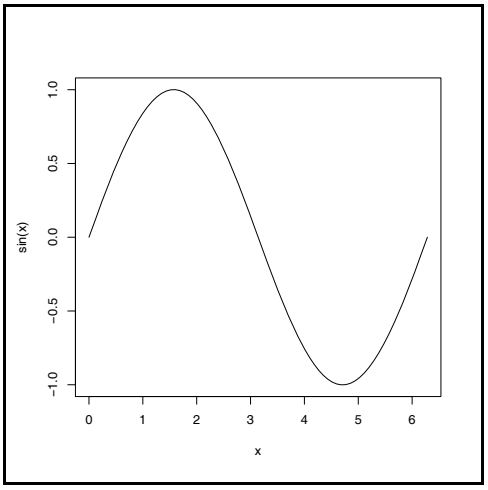
کُدهای زیر توجه کنید.

```
> set.seed(1)
> x <- rnorm(10000)
> x[x < -3.5 | x > 3.5] <- NA
> hist(x, freq=F, breaks=30)
> curve(dnorm(x), -3.5, 3.5, add=T)
```

شکل ۲-۹ آن را نشان می‌دهد.

۲-۱-۲ توابع نموداری آماری

در زبان R می‌توان از توابع نمودار آماری استفاده نمود، که کاربرد فراوانی دارد.



شکل ۲-۷: نمایش تابع $\sin(x)$

hist(x): این تابع هیستوگرام ایجاد می‌کند.

qqnorm(x): این تابع چندک‌ها را روی دو محور ایجاد می‌کند. که چندک‌های نرمال روی محور xها قرار دارد.

qqplot(x,y): این تابع چندک x را بر حسب y رسم می‌کند.

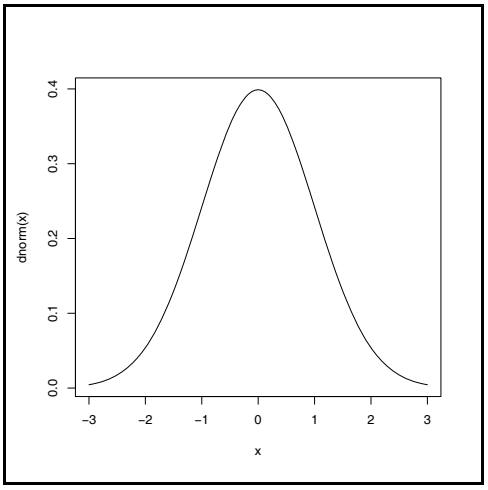
boxplot(x): این تابع نمودار box & whisker را ایجاد می‌نماید.

اکنون به مثال زیر توجه کنید (شکل ۲-۱۰).

```
> x <- rnorm(100)
> y <- rt(100, df=3)
> par(mfrow=c(2,2))
> hist(x, col=2)
> qqnorm(x)
> qqplot(x,y)
> boxplot(x, col="green")
```

۲-۱-۳ قسمت‌های گوناگون صفحه ترسیم

یک نمودار شامل سه ناحیه است. ناحیه ترسیم که توسط ناحیه نمودار احاطه شده است و این ناحیه نیز به نوبه خود به‌وسیله چهار حاشیه احاطه گردیده است. شکل ۲-۱۱ این مطلب را به‌خوبی نشان می‌دهد. حواشی خارجی با پارامتر oma تنظیم می‌گردد که مقادیر پیش‌فرض آن‌ها برابر صفر است. حواشی که ناحیه ترسیم را احاطه نموده است با پارامتر mar تنظیم می‌شود.



شکل ۲-۸: نمایش تابع نرمال استاندارد

۲-۲ وضعیت چند نمودار نسبت به هم

در زبان R، با هر دستور ترسیم high level یک پنجره جدید باز می‌شود که پنجره قبلی را می‌بندد. بنابراین نمودار قبلی از بین می‌رود. برای این که چند نمودار را تواماً داشته باشیم سه راه حل را می‌توان پیش روی داشت.

۱) ایجاد چند پنجره که یکی دیگری را از بین نبرد.

۲) ترسیم نمودارهای جداگانه بر روی یک صفحه بدون آنکه آنها روی هم قرارگیرند.

۳) ترسیم نمودار جدید بر روی نمودار قبلی در صفحه واحد، بدون آنکه نمودار قبلی از بین برود.

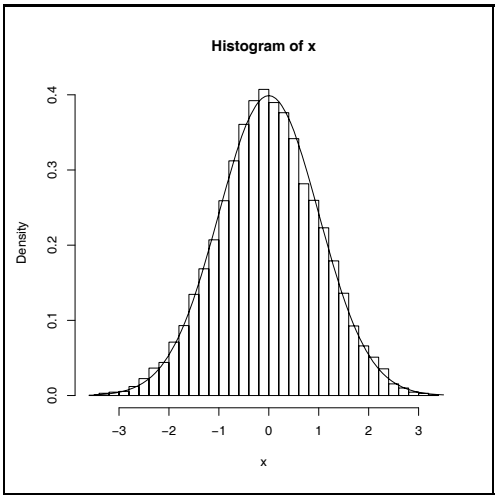
۲-۲-۱ ایجاد چند پنجره مجزا

با تابعی به نام ()windows می‌توان این کار انجام داد.

```
windows(width, height)
```

که در آن آرگومان width اندازه محور طول‌ها و آرگومان height اندازه محور عرض‌ها را نشان می‌دهد. با هر بار اجرای دستور windows یک پنجره جدید گشوده می‌شود و آماده است که در آن یک نمودار جدید ترسیم گردد. به‌عنوان مثال به کُدهای زیر توجه کنید.

```
> x <- c(2,3.3,4,5.5,6.5,8,9.5,10.10,12,13.13)
> y <- seq(2,20,by=2)
> n <- 2.75
> windows(n,n)
> plot(x,y)
> windows(n,n)
> plot(x,y,typ="o")
```



شکل ۲-۹: نمایش تابع نرمال استاندارد و هیستوگرام مربوط به آن

```
> windows(n,n)
> plot(x,y,typ="h")
> windows(n,n)
> plot(x,y,typ="s")
```

با اجرای کدهای زیر شکل ۲-۱۲ حاصل می‌گردد.

۲-۲-۲ بستن پنجره‌های باز شده نمودارها

هر بار که تابع windows() اجرای می‌گردد یک پنجره جدید گشوده می‌شود. ذکر این نکته ضروری است که شماره پنجره در زبان R از شماره دو شروع شده و شماره یک وجود ندارد. اکنون برای بستن هر از آنها می‌توان دو کار را انجام داد.

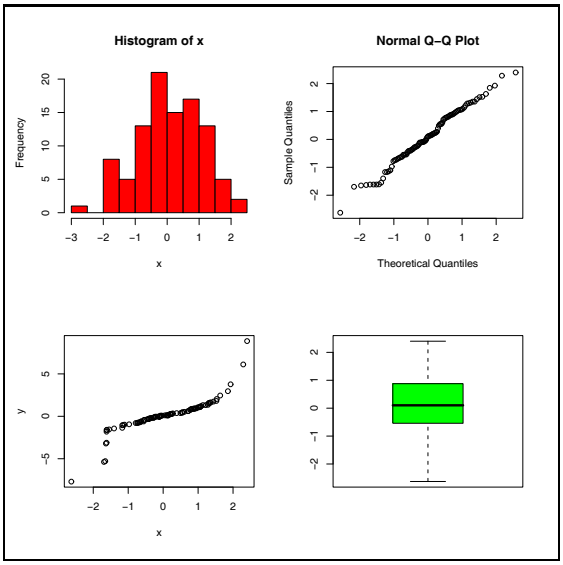
- کلیک کردن روی علامت close صفحه windows مربوطه
- استفاده از تابع dev.off() که به‌صورت زیر است.

```
> dev.off(number of window)
```

اگر آرگومان شماره پنجره داده نشود. پیش‌فرض این است که کلیه پنجره‌های موجود بسته می‌شوند.

۲-۲-۳ چند نمودار مجزا روی یک صفحه

در زبان R برای ترسیم چند نمودار مجزا روی یک صفحه نمودار روش‌های مختلفی وجود دارد که به شرح زیر ارائه می‌گردد.



شکل ۲-۱۰: نمایش چند نمودار آماری

۲-۲-۳-۱ تقسیم صفحه نمودار به قسمت‌های مساوی

با استفاده از پارامترهای mfrow و یا mfcol می‌توان روی یک صفحه چند نمودار را قرار داد. mf مخفف واژه multiframe است و row سطر و col ستون را بیان می‌کنند. تابع par() پارامترهای نمودار را تنظیم می‌کند که شرح مفصل آن در همین فصل خواهد آمد. هر دو پارامتر به شرح زیر تنظیم می‌شوند.

```
par(mfrow=c(r,k))
par(mfcol=c(r,k))
```

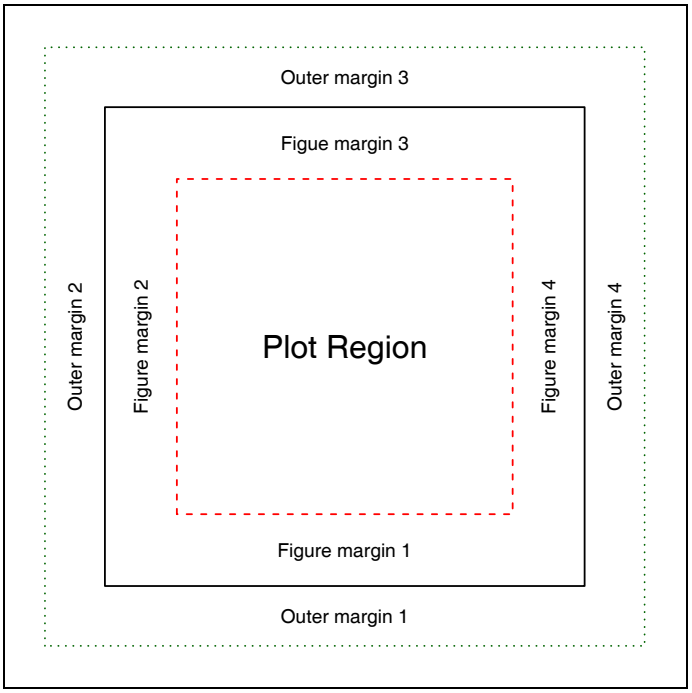
که در آن r تعداد سطرها و k تعداد ستون‌های صفحه‌ای که در آن نمودار رسم می‌گردد را نشان می‌دهد. پارامتر گرافیکی mfrow بیانگر چیدمان سطری و mfcol بیانگر چیدمان ستونی است.

۲-۲-۳-۲ تقسیم صفحه نمودار به قسمت‌های نامساوی

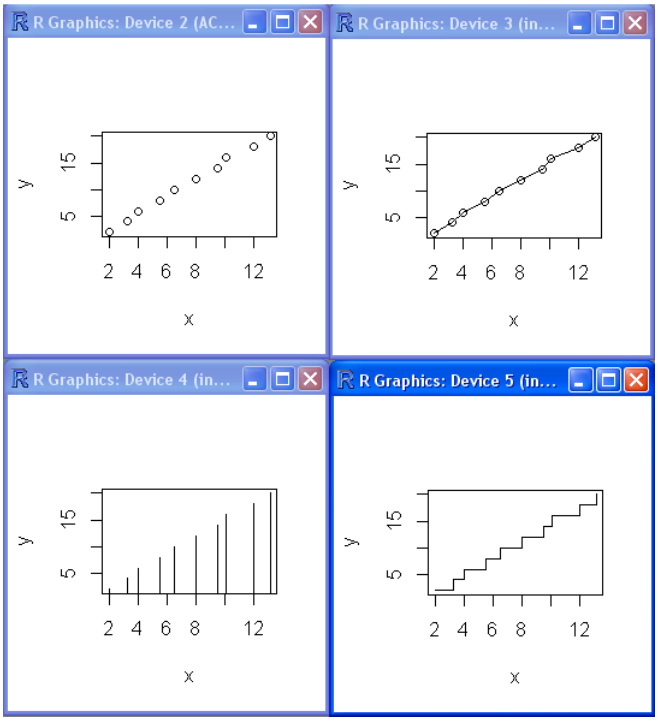
تابع layout() روش جایگزینی برای پارامترهای mfrow و mfcol است. اختلاف اولیه آنها این است که تابع اخیر می‌تواند صفحه نمایش نمودار را به چند ناحیه نامساوی تقسیم می‌کند. به عبارت دیگر در حین تقسیم‌بندی صفحه نمایش نمودار به چند سطر و ستون، ارتفاع سطرها و پهنای ستون‌ها به‌طور مجزا قابل کنترل است و یک نمودار مشخص می‌تواند بیش از یک سطر و یا ستون را اشغال نماید.

اولین آرگومان (که اجباری است) تابع layout()، تابع ماتریس است. محتوای ماتریس اعداد صحیح است که تعداد سطر و ستون اشغال شده توسط نمودارها را معین می‌کند. عدد صفر نیز در هر یک از درایه‌های ماتریس، نشانه آن است که در آنجا نموداری قرار نمی‌گیرد.

برای مثال دستور par(mfrow=c(3,2)) معادل دستور زیر است.



شکل ۲-۱۱: نمایش بخش‌های نمودار



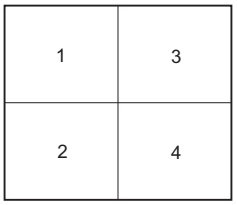
شکل ۲-۱۲: نمایش چند پنجره توام

```
layout(matrix(c(1,2,3,4,5,6), byrow=T, ncol=2))
```

می‌توان افرازهایی که تابع layout() ایجاد می‌کند را با تابع layout.show() به نمایش گذاشت. برای تابع layout() می‌توان مثال‌های بیشتری را ارائه نمود. در این مثال‌ها نحوه تقسیم‌بندی بهتر مشخص می‌گردد و ضمناً با تابع layout.show() می‌توان تقسیم‌بندی ایجاد شده را به نمایش گذاشت.

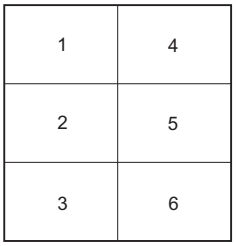
مثال ۱:

```
> m <- matrix(1:4, ,2, 2)
> m
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> layout(m)
> layout.show(4)
```



مثال ۲:

```
> m <- matrix(1:6, 3, 2)
> m
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> layout(m)
> layout.show(6)
```



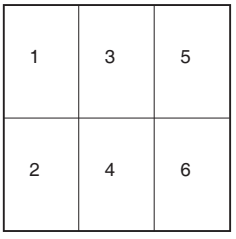
محتوی ماتریس تابع layout() ترتیب قرارگرفتن نمودارها را در ناحیه شکل‌ها نشان می‌دهد. برای مثال اگر مورد اخیر به‌صورت زیر نوشته شود.

```
layout(matrix(c(6:1), byrow=T, ncol=2))
```

ترتیب قرارگرفتن نمودارها در صفحه نمایش، به صورت عکس خواهد بود. یعنی اولین نمودارکه قبلاً در ناحیه 1×1 قرار می‌گرفت اکنون در ناحیه 2×3 قرار می‌گیرد.

مثال ۳:

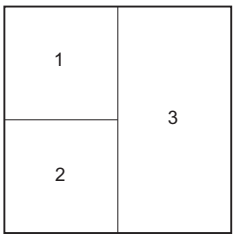
```
> m <- matrix(1:6, 2, 3)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> layout(m)
> layout.show(6)
```



مثال ۴:

یک نمودار ممکن است بیش از یک سطر و یا ستون را اشغال کند. برای مثال، اگر کدهای layout به صورت زیر است

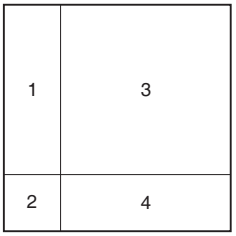
```
> m <- matrix(c(1,2,3,3), byrow=F,
                  ncol=2)
> m
      [,1]  [,2]
 [1,]    1     3
 [2,]    2     3
> layout(m)
> layout.show(3)
```



مثال ۵:

به‌صورت پیش‌فرض، تمام سطرها دارای ارتفاع یکسان و تمام ستون‌ها دارای عرض یکسان هستند. اما عرض‌ها و ستون‌های نواحی را می‌توان عوض نمود. آرگومان heights می‌تواند ارتفاع سطرهایی که به آنها دسترسی پیدا می‌کند را به نسبتی بزرگتر کند. همچنین آرگومان width مشابه آرگومان قبلی برای عرض‌های قابل دسترس عمل می‌کند.

```
> m <- matrix(1:4, 2, 2)
> layout(m, widths=c(1, 3),
+ heights=c(3, 1))
> layout.show(4)
```



پاره‌ای ارتفاع‌ها و عرض‌ها به نسبت $\frac{3}{1+3} = \frac{3}{4}$ بزرگتر می‌شوند و پاره‌ای از آنها به نسبت $\frac{1}{1+3} = \frac{1}{4}$ کوچکتر می‌گردند.

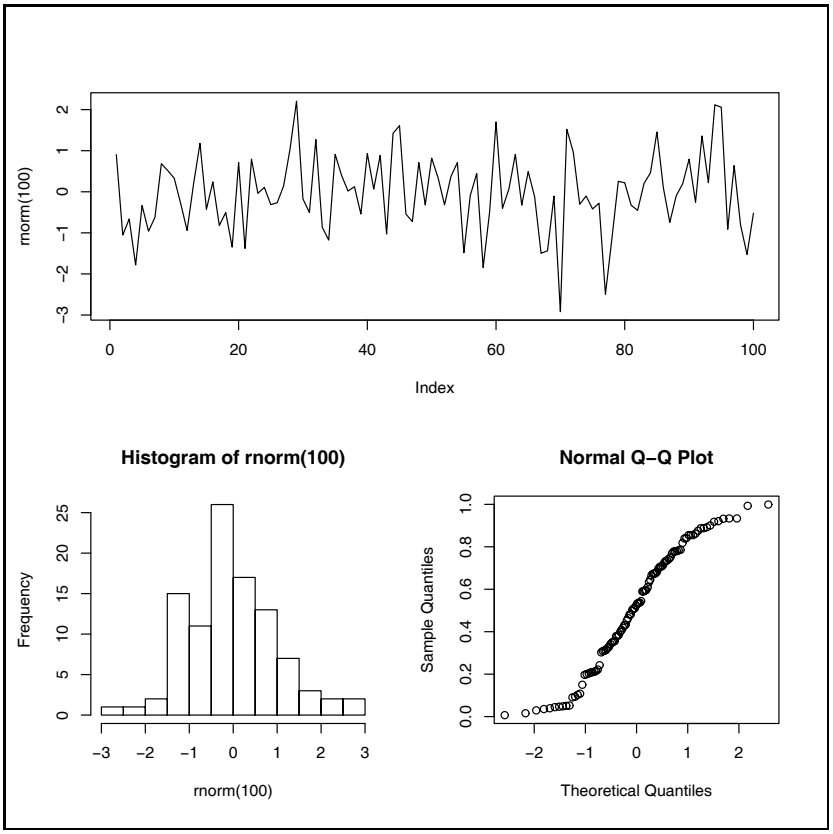
مثال ۶:

```
> layout(matrix(c(1,2,1,3),2,2))
> plot(rnorm(100), type="l")
> hist(rnorm(100))
> qqnorm(runif(100))
```

با اجرای دستورات فوق شکل ۲-۱۳ نتیجه می‌شود.

۲-۲-۳-۳ تقسیم صفحه با تعیین مکان نمودارها

همان‌طور که ملاحظه شد می‌توان توسط دستورات mfrow و mfcol به‌عنوان آرگومان par() چند نمودار را روی یک صفحه نمودار ترسیم نمود. اما امکان دیگری نیز هست که کنترل بیشتری را روی محل قرار گرفتن



شکل ۲-۱۳: نمایش چند نمودار آماری با تقسیم‌بندی نامساوی

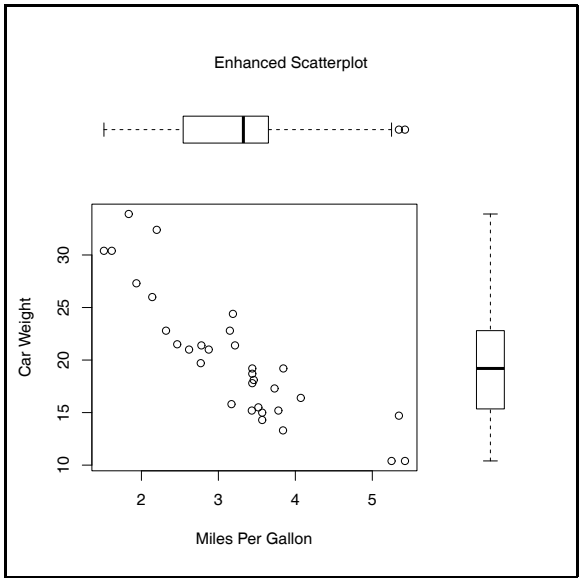
نمودار اعمال می‌کند و آن آرگومان fig در par() می‌باشد.
برای تنظیم محل نمودار، باید در آرگومان fig از فرمت زیر استفاده شود.

```
c(x_left, x_right, y_left, y_right)
```

که در آن، هر یک از مؤلفه‌های مذکور بین مقادیر صفر و یک قرار دارند. اکنون به مثال زیر توجه کنید.

```
# Add boxplots to a scatterplot
par(fig=c(0,0.8,0,0.8), new=TRUE)
plot(mtcars$wt, mtcars$mpg, xlab="Miles Per Gallon",
ylab="Car Weight")
par(fig=c(0,0.8,0.55,1), new=TRUE)
boxplot(mtcars$wt, horizontal=TRUE, axes=FALSE)
par(fig=c(0.65,1,0,0.8),new=TRUE)
boxplot(mtcars$mpg, axes=FALSE)
mtext("Enhanced Scatterplot", side=3, outer=TRUE, line=-3)
```

با اجرای برنامه فوق شکل ۲-۱۴ حاصل می‌گردد.



شکل ۲-۱۴: نمایش چند نمودار توسط دستور fig

۲-۲-۴ ترسیم نمودار جدید بر روی نمودار قبلی در صفحه واحد

فرض کنید با تابع ()plot یک نمودار ترسیم شده است. با چهار تابع می‌توان روی نمودار فعلی نمودارهای دیگری را به آن اضافه نمود.

```
plot()          # draw a graph
points()        # adds points
lines()         # adds lines
abline(a,b)     # Draws a line of slope a and intercept b
    abline(h=) # adds horizontal lines
    abline(v=) # adds vertical lines
curve()         # adds curves
```

۲-۲-۴-۱ تنظیم‌های خارجی

فهرست زیر، پارامترهای بیشتر و جزیی‌تر از نمودارها را نشان می‌دهد. برای مثال plot(x,y, col=2)

lwd: با این پارامتر می‌توان ضخامت خطوط نمودار را تعیین کرد.

lty: با این پارامتر می‌توان نوع خطوط مورد استفاده در نمودار را را تعیین کرد. مقدار این پارامتر می‌تواند عدد و یا کاراکتر باشد. برای مثال lty="dashed"

col: با این پارامتر می‌توان رنگ نمودار تعیین کرد. مقدار این پارامتر می‌تواند عدد و یا کاراکتر باشد. برای مثال col = "red". برای تغییر رنگ عنوان و زیرعنوان از col.main و col.sub، برای تغییر رنگ عناوین محورها از col.lab و بالاخره برای تغییر رنگ محورها از col.axis استفاده می‌شود.

font: مقدار عددی این پارامتر قلمِ نوشته‌های نمودار را معین می‌کند.

pch: با پارامتر pch[3] می‌توان نشانه‌های نمودار را معین نمود. مثلاً نشانه دایره به مربع تبدیل شود.

main: با این پارامتر می‌توان عنوان اصلی نمودار را مشخص کرد. فونت و اندازه آن با font.main و رنگ آن با col.main ویرایش می‌گردد.

sub: با این پارامتر می‌توان عنوان فرعی نمودار را مشخص کرد. فونت و اندازه آن با font.sub و رنگ آن با col.sub ویرایش می‌گردد.

xlab, ylab: با این پارامتر می‌توان اسامی محورهای مختصات را مشخص کرد. البته در پاره‌ای از توابع سطح بالا، این عمل خودبه‌خود انجام می‌شود.

xlim, ylim: با این پارامتر می‌توان مقادیر حداقل و حداکثر محورهای x و y را تعیین نمود. برای این کار می‌توان از دستورات xlim=c(low,high) و ylim=c(low,high) استفاده نمود، که در آن low مقدار حداقل و high مقدار حداکثر را نشان می‌دهد.

cex: با این پارامتر می‌توان نشانه‌ها و متون نمودار را بزرگتر نمود. اگر بخواهید اندازه عنوان را بزرگ کنید از cex.main و برای زیرعنوان از cex.sub استفاده کنید. برای بزرگتر نمودن اسامی محورها از cex.lab می‌توان کمک گرفت.

پاره‌ای از پارامترهای گرافیکی می‌توانند به‌صورت برداری تنظیم شوند به‌طوری که هر نقطه، متن و یا نشانه‌ای شامل عناصر بردار می‌گردد. اما به‌صورت جداگانه نیز می‌توان عمل نمود.

۲-۲-۴-۲ تنظیم پارامترهای یک نمودار

همان‌طورکه تاکنون مشاهده شد. یک نمودار می‌تواند دارای پارامترهایی نظیر ضخامت خطوط، نوع خط، رنگ، فونت، اندازه و . . . باشد. اکنون اگر پارامترهای گرافیکی در خود دستور ترسیم آورده شود، فقط برای همان دستور قابل اعمال است. اما اگر پارامترهای گرافیکی در تابع par() آورده شود، این پارامترها روی تمام نمودارهای بعدی نیز اعمال می‌گردد. مگر اینکه یا به تابع par() بعدی برسد و یا داخل یک دستور ترسیم پارامترها به‌صورت محلی تعویض گردد. برای درک بهتر مطلب به مثال‌های زیر توجه کنید.

```
> plot(x,y,cex.lab=2, main="size of title",cex.main=2,cex.axis=2)
```

در مورد فوق مشاهده می‌شود که پارامترهای شکل داخل تابع plot() قرار دارد.

```
> par(cex.lab=2,cex.main=2,cex.axis=2)
> plot(x,y,main="size of title")
```

تعویض محلی به‌صورت زیر است.

```
> par(cex.lab=2,cex.main=2,cex.axis=2)
> plot(x,y,main="size of title")
```

3. Plotting character

```
> plot(x,y,cex.lab=2, main="size of title",cex.main=1,cex.axis=1)
```

اکنون به تعویض‌های جالب تابع par() در مثال زیر توجه کنید.

```
> # Set a graphical parameter using par()

> #par()               # view current settings
> opar <- par()        # make a copy of current settings
> par(col.lab= "red") # red x and y labels
> hist(mtcars$mpg)    # create a plot with these new settings
> par(opar)           # restore original settings
```

همان‌طور که تاکنون مشاهده شد تابع par() نقش زیادی در ویرایش نمودارها ایفاء می‌کند. بنابراین به نکات دیگری در این باب پرداخته می‌شود. اندازه ناحیه یک شکل تقریباً دارای 6 اینچ پهنا و 6 اینچ درازا است. شما می‌توانید این امر را در R به محک بزنید.

```
> par("fin")
[1] 5.781249 5.770832
```

آرگومان mar حواشی نمودار را معین می‌کند. در واقع صورت کلی آن به‌صورت زیر است.

```
> par(mar=c(bottom, left, top, right))
```

برای به‌دست آوردن مقادیر پیش‌فرض آن، باید دستور زیر را اجرا نمود.

```
> par("mar")
[1] 5.1 4.1 4.1 2.1
```

بنابراین مقدار آرگومان به‌صورت c(5.1, 4.1, 4.1, 2.1) و یا c(5, 4, 4, 2)+0.1 است. واحد آنها به اندازه خط متن است. کد زیر با استفاده از تابع mtext() خطوط حاشیه به‌صورت برچسب را مشخص می‌کند.

```
plot(1:10, ann=FALSE, type="n", xaxt="n", yaxt="n")
for(j in 1:4) for(i in 0:10) mtext(as.character(i),side=j,line=i)
```

از اجرای کد بالا شکل ۲-۱۵ حاصل می‌گردد.

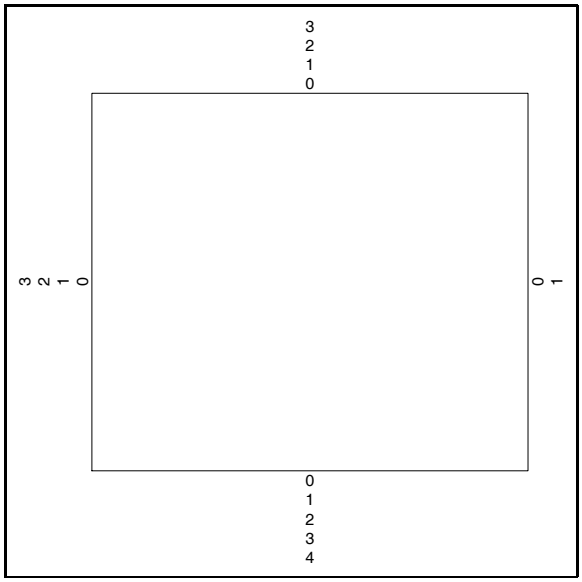
راه دیگری برای تعیین حواشی نمودارها موجود است و با استفاده از آرگومان mai انجام می‌شود و بر حسب اینچ می‌باشد. برای به‌دست آوردن مقادیر پیش‌فرض آن، باید دستور زیر را اجرا نمود.

```
> par("mai")
[1] 1.02 0.82 0.82 0.42
```

در هر دو آرگومان بالا یعنی mar و mai بزرگترین مقادیر متعلق به طرف پایین، چپ و بالا است، که در آنها عناوین نمودار و محورها قرار می‌گیرد. اگر نسبت این دو را حساب کنید اعداد 0.2 اینچ حاصل می‌گردد.

```
> par("mai")/par("mar")
[1] 0.2 0.2 0.2 0.2
```

آرگومان mgp در تابع par() برداری است که دارای سه مقدار است و بر حسب خطوط عناوین حاشیه شکل می‌باشد و فاصله عنوان نمودارها را تا اعداد آن معین می‌کند. پیش‌فرض آن c(3,1,0) است.



شکل ۲-۱۵: نمایش خطوط حاشیه به‌صورت برچسب

```
> par("mgp")
[1] 3 1 0
```

آرگومان oma(outer margin area) برداری است که چهار عنصر پایین، چپ، بالا و راست را دارا می‌باشد. این آرگومان اندازه حاشیه خارجی را بر حسب خطوط متن به‌دست می‌دهد و پیش‌فرض آن c(0,0,0,0) است.

```
> par("oma")
[1] 0 0 0 0
```

برای مشخص‌تر شدن حاشیه خارجی یک نمودار به شکل ۲-۱۶ توجه کنید.

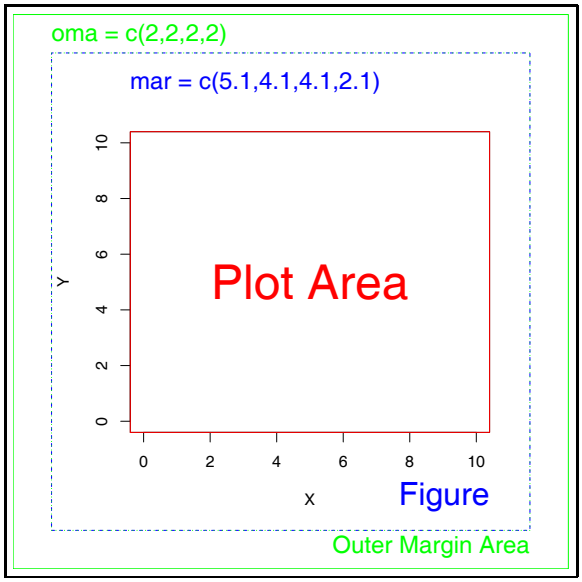
۲-۲-۴-۳ پاره‌ای از توابع سطح پایین

شما یک نمودار ایجاد می‌کنید، سپس می‌خواهید که بعضی از موارد را به آن اضافه کنید. این کار توسط توابع سطح پایین امکان‌پذیر است.

افزودن خطوط

توابع lines() و abline() برای افزودن خطوط به نمودار موجود استفاده می‌شود. تابع lines() نقاط بردار ورودی را به هم وصل می‌کند. تابع abline() خطوط راست با شیب و عرض از مبداء معین را ترسیم می‌کند. به مثال زیر توجه کنید.

```
> plot(c(-2,2),c(-2,2))
> lines(c(0,2), c(0,2), col="red")
> abline(a=1, b=2, lty=2) # adds the line y = a + bx
```



شکل ۲-۱۶: نمایش حاشیه خارجی نمودار

```
> abline(v=1, lty=3, col="blue", lwd=3)
```

با اجرای دستورات فوق نمودار زیر بدست می‌آید. در تابع abline() با پارامتر v خط عمودی و با پارامتر h خط افقی رسم می‌گردد.

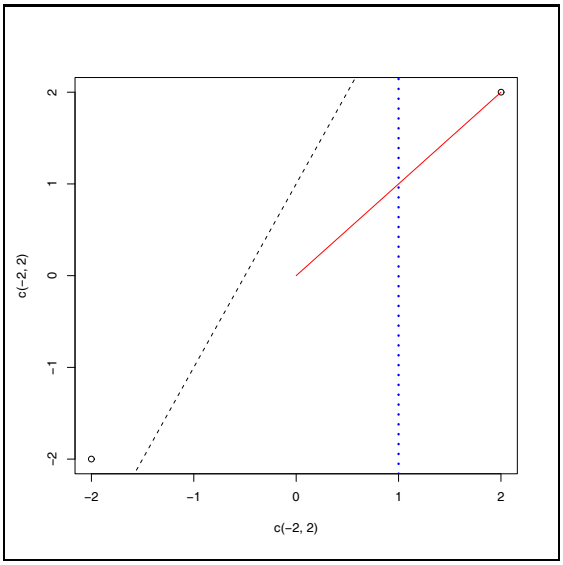
توابع arrows() و segments() به ترتیب برای ترسیم پیکان و پاره خط به‌کار می‌رود. به عنوان مثال به دستورات قبل دستورات زیر اضافه می‌شود.

```
> arrows(
+ c(0,0,0),
+ c(1,1,1),
+ c(0,0.5,1),
+ c(1.2,1.5,1.7),
+ length = 0.1)
```

به نمودارها می‌توان خطوط راهنما[۴] افزود. برای این کار از تابع grid() استفاده می‌شود. این تابع مانند تابع plot() دارای آرگومان‌های رنگ، ضخامت، نوع خط (خط‌پر، خط‌چین و ...) می‌باشد. گرامر این آرگومان‌ها مانند تابع plot() است. تابع grid() دو آرگومان nx و ny وجود دارد که بود و نبود خطوط راهنما را به ترتیب در محورهای x و y معین می‌کنند. اگر هر یک از آنها مساوی NA گردد، خطوط در جهت‌های x و یا y رسم نمی‌گردد. اکنون فرض کنید که در جهت محور y خطوط راهنما می‌خواهید و در جهت محور x به خطوط راهنما نیاز ندارید، آنگاه باید به‌صورت زیر عمل نمود.

```
grid(nx=NA, ny=NULL)
```

4. Grid lines



شکل ۲-۱۷: نمایش چند خط راست

خطوط راهنما در دو جهت x و y به تعداد تیک‌های محورهای مربوط رسم می‌شوند، اما اگر مایل باشید که تعداد آنها را افزایش و یا کاهش دهید اولین روش این است که تیک‌ها را به تناسب در نمودار مورد نظر تغییر دهید. اما روش دوم استفاده از تابع abline(h=numeric, v=numeric) می‌باشد. اکنون به مثال‌های زیر توجه کنید.

```
> x <- c(1,4,7,8,10)
> y <- c(2,6,11,15,20)
> plot(x,y)
> grid()
```

با اجرای کُدهای فوق نمودار شماره ۲-۱۹ ایجاد می‌گردد. اکنون خطوط راهنما به‌صورت غیر پیش‌فرض (سفارشی) اعمال می‌گردد.

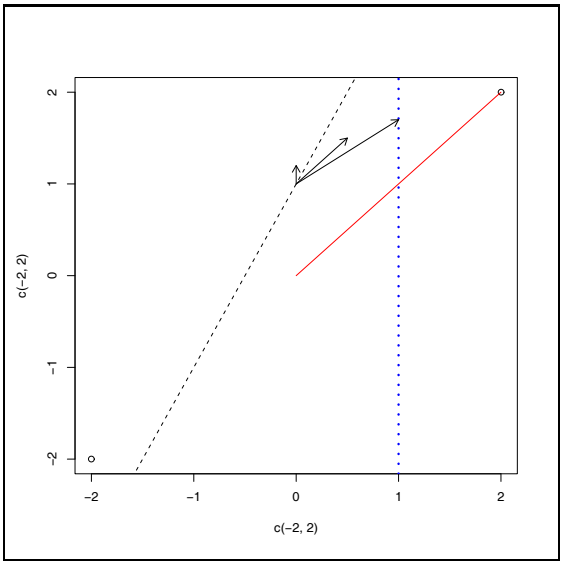
```
> x <- c(1,4,7,8,10)
> y <- c(2,6,11,15,20)
> plot(x,y)
> abline(h=seq(1,20,2),v=seq(1,10,0.5),lty=3,col="gray50")
```

با اجرای کُدهای فوق نمودار شماره ۲-۲۰ ایجاد می‌گردد.

افزودن نقاط و نشانه‌ها

تابع points() نقاط و نشانه اضافی را به نمودار موجود می‌افزاید. کُدهای زیر نقاطی را به نمودار قبلی (شکل ۲-۲۱) اضافه می‌کند.

```
> points(rnorm(4), rnorm(4), pch=3, col="blue")
```



شکل ۲-۱۸: نمایش چند پیکان

```
> points(rnorm(4), rnorm(4), pch=4, cex=3, lwd=2)
> points(rnorm(4), rnorm(4), pch="K", col="green")
```

برنامه زیر فهرستی از نشانه‌هایی که می‌توان مورد استفاده قرار داد را نشان می‌دهد.

```
> plot(0:10, 0:10, type="n", xlab="", ylab="")
> k <- -1
> for (i in c(2,5,8)) {
+ for (j in 0:9) {
+ k <- k+1
+ points(i, j, pch=k, cex=2)}
+ }
```

با اجرای برنامه بالا شکل ۲-۲۲ ایجاد می‌گردد.

افزودن عنوان و متن

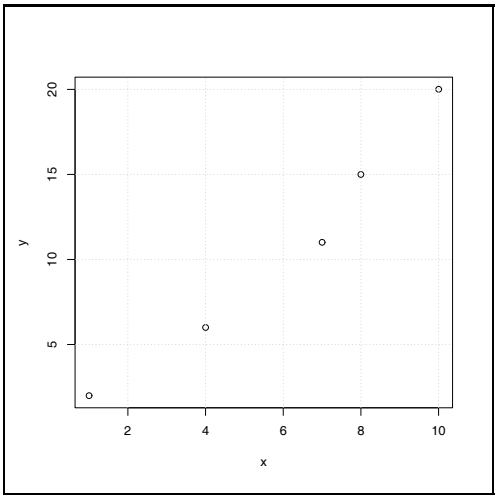
توابع title()، legend()، mtext() و text() به نمودار موجود (فقط در ناحیه ترسیم) متونی را اضافه می‌نمایند. کُدهای زیر متونی (شکل ۲-۲۳) را به نمودار قبلی اضافه می‌کند.

```
> title(main="My title", sub="My subtitle")
> text(0,0, "some text")
> text(1,1, "Angular text", srt=45)
```

تابع mtext()[5] متن مورد نظر را به یکی از حواشی نمودار موجود (شکل ۲-۲۴) اضافه می‌کند.

```
mtext("Text in the margin", side=4)
```

5. marginal text



شکل ۲-۱۹: نمایش یک نمودار با خطوط راهنما

side: محور مورد نظر را تعیین می‌کند. عدد 1 برای آن محور پایین، عدد 2 محور چپ، عدد 3 محور بالا و بالاخره عدد 4 محور راست را مشخص می‌کند.

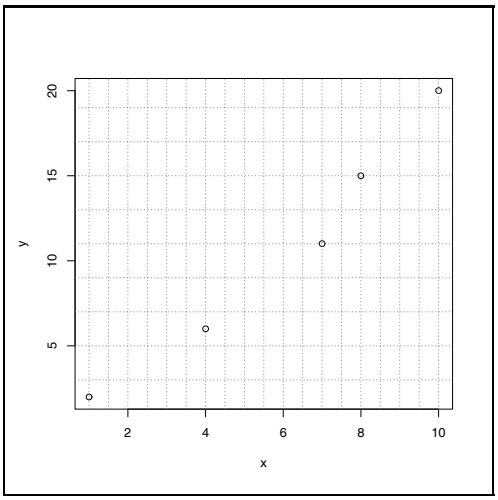
line: فاصله نوشته را (از محور مورد نظر) معین می‌کند. پیش‌فرض آن 0 است.

at: مختصات نوشته را روی خط فوق نشان می‌دهد. اگر بخواهید روی یک نمودار فرمول‌های ساده و الفبای یونانی را بنویسید از تابع ()expression استفاده کنید. دستورات آن شبیه دستورات LaTeX است. برای یافتن فهرست کامل آن‌ها در محیط R، دستور (demo(plotmath را تایپ کنید. اکنون به کُدهای زیر توجه کنید، که به نمودار (شکل ۲-۲۵) قبل اضافه می‌شود.

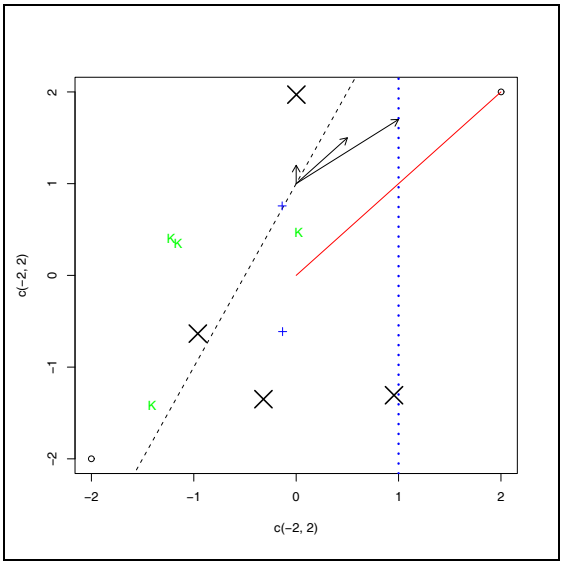
```
> text(-1,1.5,
+ expression(
+ paste(
+ frac(1, sigma*sqrt(2*pi)),
+ " ",
+ plain(e)^frac(-(x-mu)^2, 2*sigma^2))),
+ cex = 1.2)
```

برای مثال چند نمونه از فرمول‌ها که توسط کُدهای زیر نوشته شده است ارائه می‌گردد، که خروجی آنها نیز در شکل ۲-۲۶ قابل مشاهده است.

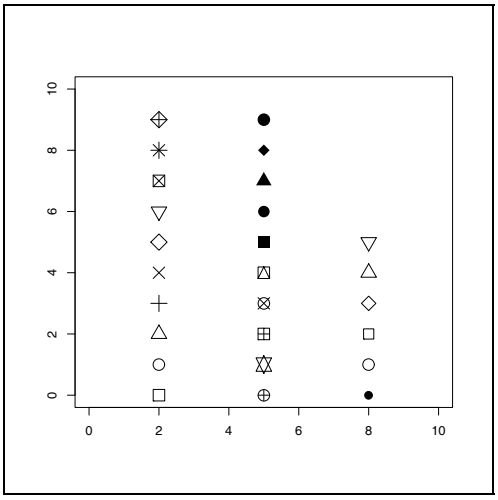
```
> par(mar = c(1, 1, 1, 1))
> plot(0:10, 0:10, type = "n", axes = FALSE)
> text(1, 10, expression(x %+-% y), cex = 1.5)
> text(1, 9, expression(x[i]), cex = 1.5)
> text(1, 8, expression(x^2), cex = 1.5)
> text(1, 7, expression(sqrt(x)), cex = 1.5)
> text(1, 6, expression(sqrt(x, 3)), cex = 1.5)
> text(1, 5, expression(x != y), cex = 1.5)
```



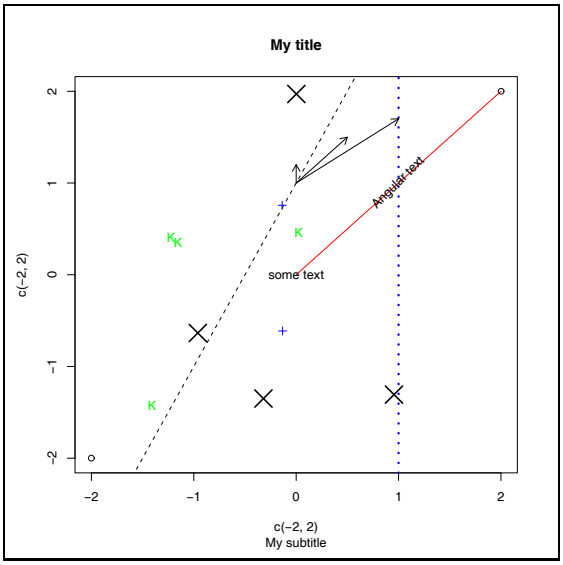
شکل ۲-۲۰: نمایش یک نمودار با خطوط راهنمای سفارشی



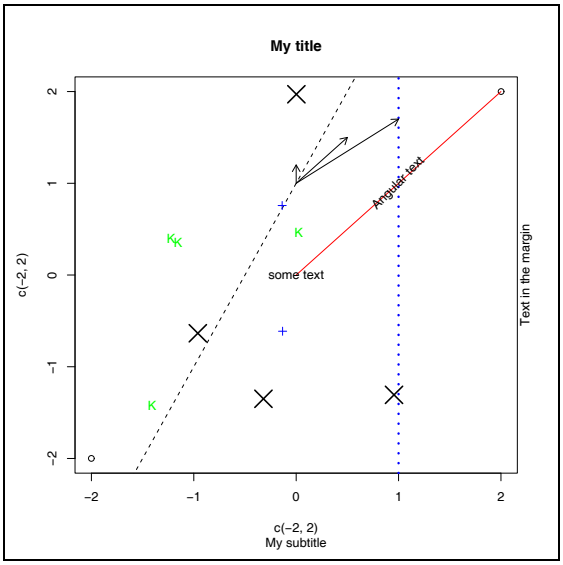
شکل ۲-۲۱: نمایش چند نقطه



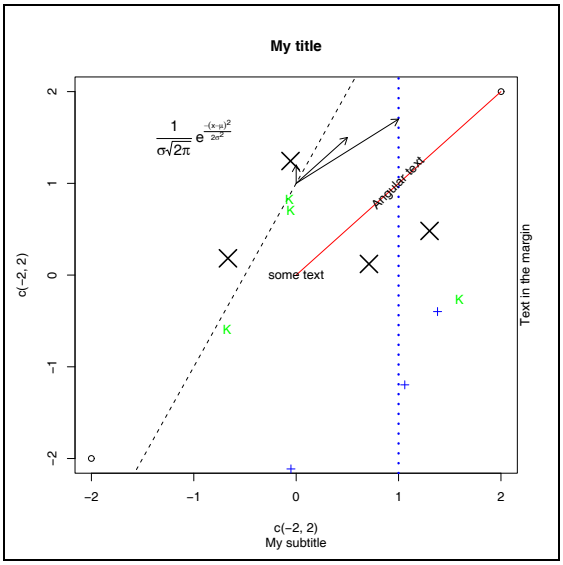
شکل ۲-۲۲: نمایش نشانه‌ها



شکل ۲-۲۳: نمایش چند متن

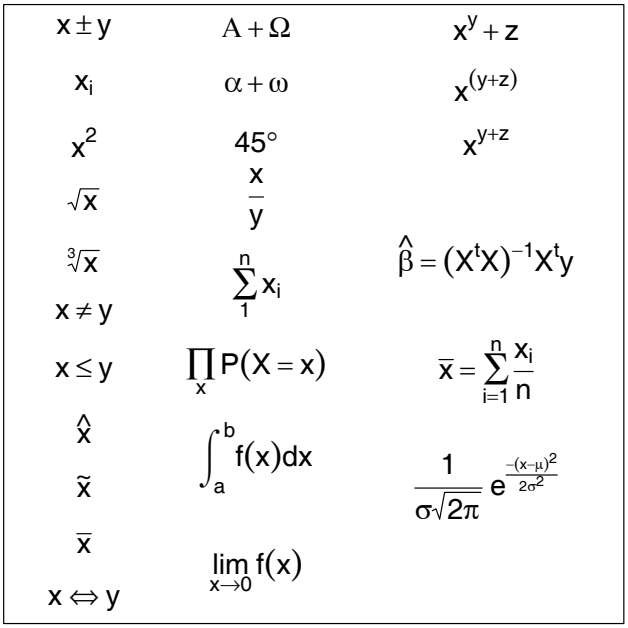


شکل ۲-۲۴: نمایش متن روی حاشیه



شکل ۲-۲۵: نمایش فرمول روی نمودار

```
> text(1, 4, expression(x <= y), cex = 1.5)
> text(1, 3, expression(hat(x)), cex = 1.5)
> text(1, 2, expression(tilde(x)), cex = 1.5)
> text(1, 1, expression(bar(x)), cex = 1.5)
> text(1, 0, expression(x %<=>% y), cex = 1.5)
> text(4, 10, expression(Alpha + Omega), cex = 1.5)
> text(4, 9, expression(alpha + omega), cex = 1.5)
> text(4, 8, expression(45 * degree), cex = 1.5)
> text(4, 7, expression(frac(x, y)), cex = 1.5)
> text(4, 5.5, expression(sum(x[i], i = 1, n)), cex = 1.5)
> text(4, 4, expression(prod(plain(P)(X == x), x)), cex = 1.5)
> text(4, 2.5, expression(integral(f(x) * dx, a, b)), cex = 1.5)
> text(4, 0.5, expression(lim(f(x), x %->% 0)), cex = 1.5)
> text(8, 10, expression(x^y + z), cex = 1.5)
> text(8, 9, expression(x^(y + z)), cex = 1.5)
> text(8, 8, expression(x^y + z), cex = 1.5)
> text(8, 6, expression(hat(beta) == (X^t * X)^{-1 + } * X^t * y), cex = 1.5)
> text(8, 4, expression(bar(x) == sum(frac(x[i], n), i == 1, n)), cex = 1.5)
> text(8, 2, expression(paste(frac(1, sigma * sqrt(2 * pi)), " ", plain(e)^{
+ frac(-(x - mu)^2, 2 * sigma^2)
+ })), cex = 1.5)
> box()
```



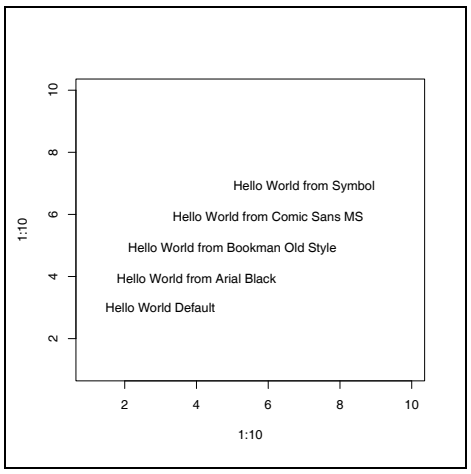
شکل ۲-۲۶: نمایش چند فرمول نوشته شده در R

تغییر فونت‌ها در نمودار

در یک نمودار می‌توان فونت‌های نوشته‌های آن را عوض نمود. برای مثال به کُدهای زیر توجه کنید.

```
plot(1:10,1:10,type="n")
windowsFonts(
A = windowsFont("Arial Black"),
B = windowsFont("Bookman Old Style"),
C = windowsFont("Comic Sans MS"),
D = windowsFont("Symbol")
)
text(3,3,"Hello World Default")
text(4,4,family="A","Hello World from Arial Black")
text(5,5,family="B","Hello World from Bookman Old Style")
text(6,6,family="C","Hello World from Comic Sans MS")
text(7,7,family="D", "Hello World from Symbol")
```

پس از اجرای کُدهای فوق شکل ۲-۲۷ را مشاهده می‌گردد.



شکل ۲-۲۷: نمایش فونت‌ها روی نمودار

کنترل محورها

وقتی شما نموداری را ایجاد می‌کنید، محورها و برچسب‌های محورها خودبه‌خود با استفاده از پیش‌فرض‌ها ایجاد می‌گردد. این تنظیمات را می‌توان با پارامترهای گرافیکی کنترل نمود. مثلاً با استفاده از axes = F محور ها حذف می‌شوند. به مثال زیر توجه کنید.

```
> x <- rnorm(100)
> y <- rnorm(100)
```

محورها خودبه‌خود رسم نمی‌شود.

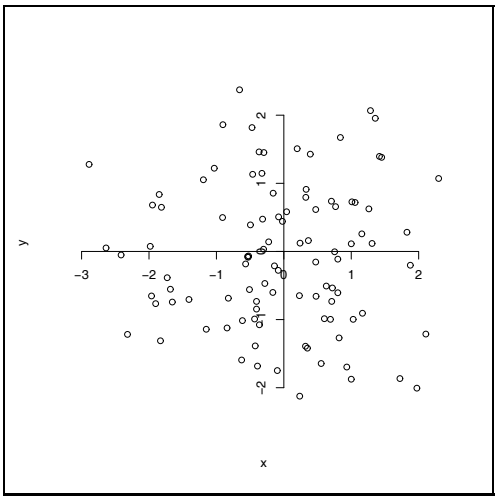
```
> plot(x,y, axes=F)
```

محورها به‌طور دستی وارد می‌شود.

```
> axis(side=1)
> axis(side=2)
```

اکنون می‌توان محورها را به کمک تابع axis() بازسازی نمود. در این تابع آرگومان side محل رسم محور را نشان می‌دهد. عدد 1 برای پایین، عدد 2 برای سمت چپ، عدد 3 برای بالا و بالاخره عدد 4 برای سمت راست. با استفاده از آرگومان pos می‌توان محل محورهای x و y را رسم نمود (شکل ۲-۲۸). به کدهای زیر توجه کنید.

```
> <- rnorm(100)
> y <- rnorm(100)
> plot(x,y, axes=F)
> axis(side=1, pos=0)
> axis(side=2, pos=0)
```



شکل ۲-۲۸: نمایش تغییر محل محورها

محل قرارگرفتن علایم تیک‌ها و برچسب‌ها روی آن‌ها با آرگومان‌های at و labels مشخص می‌گردد. شکل ۲-۲۹ آنها را نشان می‌دهد.

قرارگرفتن علایم تیک‌ها در مکان مشخص شده

```
> x <- rnorm(100)
> y <- rnorm(100)
> plot(x,y, axes=F)
> axis(side=1, pos=0)
> axis(side=2, pos=0)
```

شکل ۲-۳۰ برچسب‌ها را نشان می‌دهد.

قرارگرفتن برچسب‌ها روی علایم تیک‌ها

```
> x <- 1:20
> y <- rnorm(20)
> plot(x,y,axes=F)
```

```
> xtickplaces <- 1:20
> ytickplaces <- seq(-2,2,l=6)
> xlabels <- paste("day", 1:20, sep=" ")
> axis(side=1, at=xtickplaces, labels=xlabels)
> axis(side=2, at=ytickplaces)
```

در زبان R تمام برچسب‌های موجود نمایش داده نمی‌شود. برای این که آن‌ها روی هم قرار می‌گیرند. اگر بخواهید تمام آن‌ها روی محورها نشان داده شوند باید اندازه کاراکترها کوچک گردد (شکل ۲-۳۱). این کار با پارامتر cex.axis امکان‌پذیر است.

```
> x <- 1:20
> y <- rnorm(20)
> plot(x,y,axes=F)
> xtickplaces <- 1:20
> ytickplaces <- seq(-2,2,l=6)
> xlabels <- paste("day", 1:20, sep=" ")
> axis(side=1, at=xtickplaces, labels=xlabels, cex.axis=0.5)
> axis(side=2, at=ytickplaces)
```

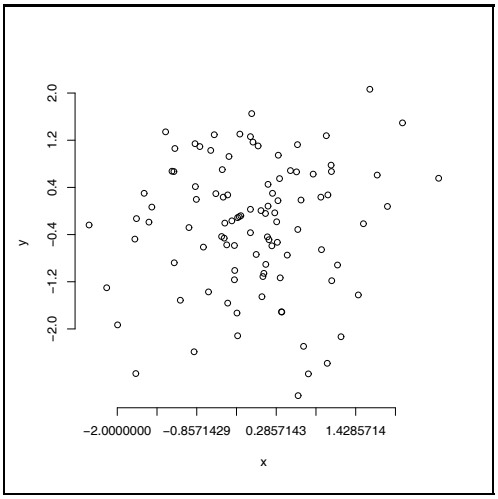
به مثال دیگری از دستورات at و labels توجه کنید. در این مثال از تابعی به نام box() استفاده شده است که پیرامون شکل کادری به اشکال متفاوت رسم می‌کند (شکل ۲-۳۲).

```
> plot(1:7, abs(rnorm(7)), type = 'h', axes = FALSE)
> axis(1, at = 1:7, labels = letters[1:7])
> box(lty = 'dotdash')
```

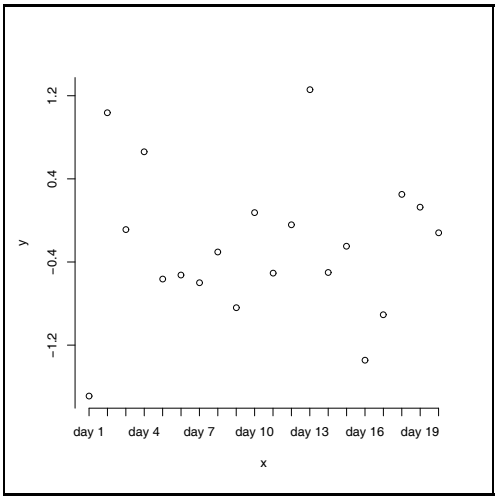
پارامتر مفید دیگری که می‌توان از آن استفاده نمود tck است. این پارامتر اندازه علامت تیک را مشخص می‌کند. حالت حدی آن tck=1 است که در واقع خط شبکه نمودار را رسم می‌کند (شکل ۲-۳۳). ضمناً پارامتر دیگری وجود دارد که tcl نامیده می‌شود که جهت تیک‌های محورهای نمودار را نشان می‌دهد. اگر مقدار آن منفی باشد تیک‌های محورها به سمت بیرون نمودار است و اگر مقدار آن مثبت باشد تیک‌های محورها به سمت داخل نمودار خواهند بود. مقدار پیش‌فرض آن tcl=-0.5 است.

```
> x <- 1:20
> y <- rnorm(20)
> plot(x,y,axes=F)
> xtickplaces <- 1:20
> ytickplaces <- seq(-2,2,l=6)
> xlabels <- paste("day", 1:20, sep=" ")
> axis(side=1, at=xtickplaces, labels=xlabels, cex.axis=0.5)
> axis(side=1, at=c(5,10,15,20,25), labels=rep("",5), tck=1, lty=2)
> axis(side=2, at=ytickplaces)
```

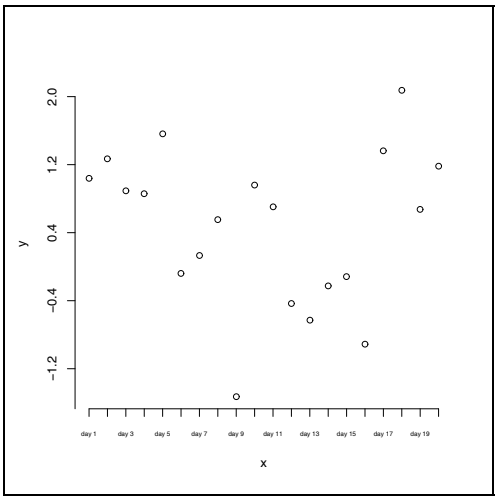
برای رسم محورهای لگاریتمی x و یا y می‌توان از log="x" یا log="y" استفاده نمود. اگر بخواهید هر دو محور لگاریتمی شود از log="xy" استفاده کنید.



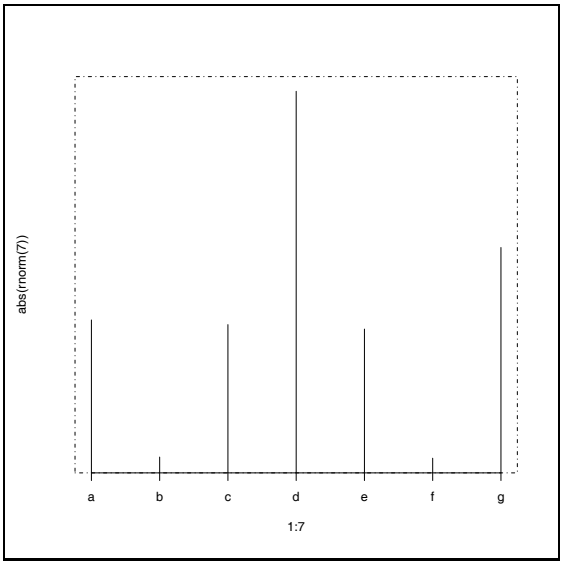
شکل ۲-۲۹: نمایش محل تیک‌ها روی محور



شکل ۲-۳۰: نمایش محل برچسب‌ها روی محور



شکل ۲-۳۱: نمایش محل برچسب‌ها روی محور با اندازه کوچک



شکل ۲-۳۲: نمایش محل برچسب‌ها روی محور به صورت حرف

نمایش تیک‌های کوچک نمودار

در ترسیم نمودارها، روی محورها درجه‌بندی‌هایی صورت می‌گیرد. این درجه‌بندی‌ها به دو بخش بزرگ[۶] وکوچک[۷] تقسیم می‌شوند. با ترسیم شکل، به‌صورت پیش فرض درجه‌بندی‌های بزرگ، خود به‌خود ایجاد می‌گردند. اما درجه‌بندی‌های کوچک به‌صورت پیش فرض صورت نمی‌گیرند. برای درج درجه‌بندی‌های کوچک در نمودار، باید از بسته Hmisc استفاده نمود. اما بسته گفته شده نیز نیاز به بسته‌های survival و splines دارد. اکنون به مثال زیر توجه کنید.

```
library(survival)
library(splines)
library(Hmisc)
plot(0:10,seq(0,20,2),xlab="m.tick of x",ylab="m.tick of y")
minor.tick(nx=2, ny=5)
```

با اجرای کُدهای فوق شکل ۲-۳۴ به‌دست می‌آید.
در par() آرگومان دیگری نیز وجود دارد که اولاً مختصات نقطه تلاقی محورها را به‌دست می‌دهد. ثانیاً محدوده ناحیه ترسیم را معین می‌کند. اگر بردار c(x1, x2, y1, y2) را به‌عنوان مقادیر usr در نظر بگیرید، آنگاه مختصات محل تلاقی محورها (x1,y1) است و محور طول‌ها از x1 تا x2 امتداد می‌یابد و محور عرض‌ها از y1 تا y2 کشیده می‌شود. اکنون به کُدهای زیر توجه کنید.

```
> plot(0, 0, type = "n", axes = FALSE, xlab = "x", ylab = "y")
> par(usr = c(1, 10, 1, 5))
> axis(side = 1, at = 1:10)
> axis(side = 2, at = seq(1,5,0.5))
> x <- c(1.5,2,3,4,5.5,6,7.5,8,9,9.5)
> y <- c(1.25,1.5,2,2.5,3,3,3.25,3.5,4,4.5)
> points(x, y)
> box()
```

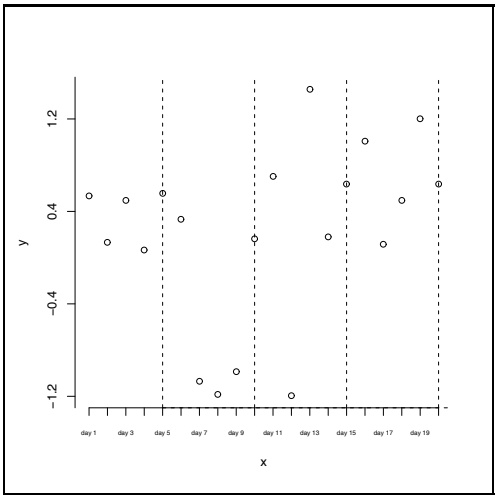
با اجرای کُدهای شکل ۲-۳۵ به‌دست می‌آید.

۲-۲-۴-۴ نمودارهای ژولیده

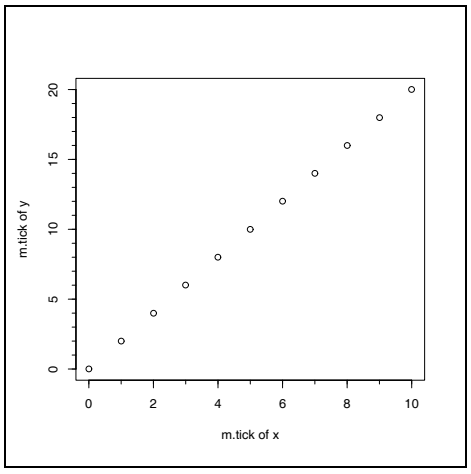
در پاره‌ای از نمودارهای پراکنش ترسیم شده، نقاط نمودار با خط به یکدیگر وصل شده‌اند. اما به سبب داده‌های آنها به‌صورت درهم برهم در می‌آیند، که صورت جالبی ندارند. در زبان R می‌توان از ترفندی استفاده نمود که این ژولیدگی نموداری رفع گردد. اکنون به مثال زیر توجه کنید.

```
> x <- c(1,2,1,1.5,4,5,1.5,4,4.5,3)
> y <- c(1,3,5,1,3,6,3.5,1,5,5)
> par(mfrow=c(1,2))
> plot(x,y,type="l", main="messy graph")
> sequence <- order(x)
```

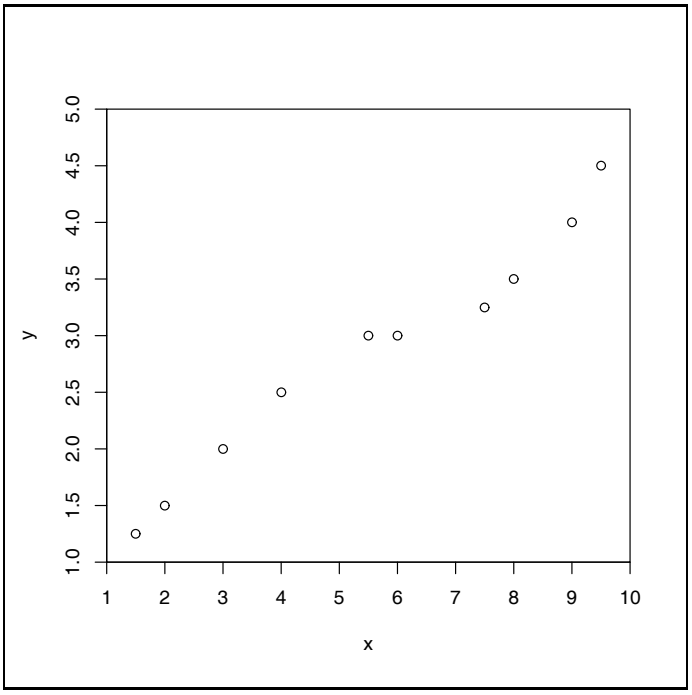
6. major tick 7. minor tick



شکل ۲-۳۳: نمایش خطوط شبکه



شکل ۲-۳۴: نمایش تیک‌های کوچک نمودار



شکل ۲-۳۵: نمایش تغییرات محورها با پارامتر usr

```
> plot(x[sequence],y[sequence],type="l",main="order graph")
```

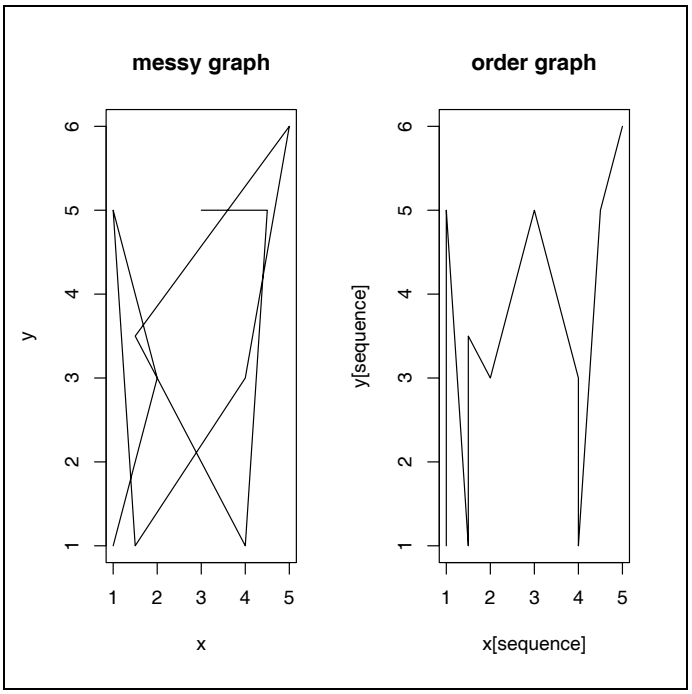
با اجرای کُدهای فوق شکل ۲-۳۶ به‌دست می‌آید. نمودار سمت چپ، نمودار ژولیده است، اما نمودار سمت راست، نمودار مرتب شده می‌باشد.

۲-۲-۴-۵ نمودارهای قطاعی

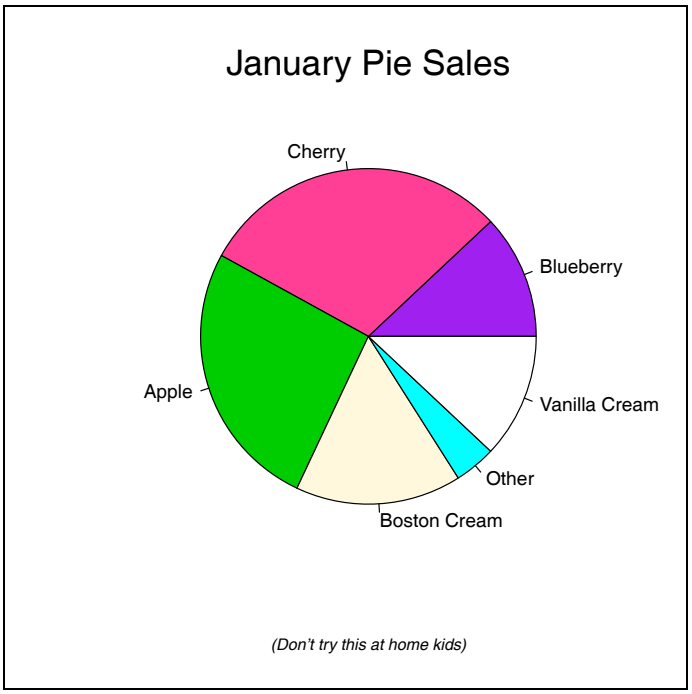
نمودار قطاعی از جمله نمودارهایی است که در نمایش‌های مختلف از آن استفاده می‌شود. برای این کار از تابع pie() استفاده می‌گردد. این تابع دارای چند آرگومان است که از همه مهم‌تر برداری است که از مقادیر غیر منفی تشکیل شده است و اندازه قطاع‌ها را نشان می‌دهد. آرگومان دوم رشته‌ای از کاراکترها است که نام قطاع‌ها را معین می‌نماید. اکنون به کُدهای زیر توجه کنید.

```
> pie.sales <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
> names(pie.sales) <- c("Blueberry", "Cherry","Apple", "Boston Cream", "Other",
                        "Vanilla Cream")
> pie(pie.sales,col = c("purple","violetred1","green3","cornsilk","cyan","white"))
> title(main="January Pie Sales", cex.main=1.8, font.main=1)
> title(xlab="(Don't try this at home kids)", cex.lab=0.8, font.lab=3)
```

با اجرای کُدهای بالا شکل ۲-۳۷ حاصل می‌گردد.



شکل ۲-۳۶: نمایش نمودارهای ژولیده و مرتب شده



شکل ۲-۳۷: نمایش نمودار قطاعی

۶-۴-۲-۲ نمودارهای سه بُعدی

در زبان R می‌توان نمودار سه بعدی نیز داشت. این کار هم با خود Rامکان‌پذیر است و هم با بسته‌های نرم‌افزاری آن می‌توان نمودار سه بُعدی ترسیم نمود. توابعی که از آنها در این خصوص می‌توان استفاده نمود به شرح زیر است.

outer() این تابع عملی مانند ضرب خارجی را انجام می‌دهد و حداقل دارای سه آرگومان است. دو آرگومان اول و دوم بردارهای x,y هستند. بردار سوم تابعی است که نسبت بین اجزاء بردارهای x,y را نشان می‌دهد. به مثال‌های زیر توجه کنید.

```
> x <- 1:3
> y <- 1:3
> z <- outer(x,y, FUN="-")
> z
```

```
     [,1] [,2] [,3]
[1,]    0   -1   -2
[2,]    1    0   -1
[3,]    2    1    0
```

```
> x <- c("A", "B", "C", "D")
> y <- 1:9
> z <- outer(x, y, paste, sep = "")
> z
```

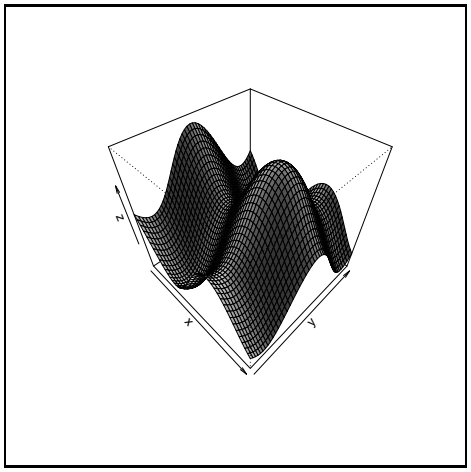
```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,] "A1" "A2" "A3" "A4" "A5" "A6" "A7" "A8" "A9"
[2,] "B1" "B2" "B3" "B4" "B5" "B6" "B7" "B8" "B9"
[3,] "C1" "C2" "C3" "C4" "C5" "C6" "C7" "C8" "C9"
[4,] "D1" "D2" "D3" "D4" "D5" "D6" "D7" "D8" "D9"
```

persp() با این تابع می‌توان یک نمای سه بُعدی و یا رویه را ترسیم نمود. اکنون به مثال زیر توجه کنید.

```
> x <- seq(-4,4,l=50)
> y <- x
> myf <- function(x,y){
+ sin(x)+cos(y)
+ }
```

```
> z <- outer(x,y, FUN = myf)
> persp(x,y,z, theta=45, phi=45, shade = 0.45)
```

با اجرای کُدهای بالا شکل ۲-۳۸ حاصل می‌شود.



شکل ۲-۳۸: نمایش یک شکل سه بُعدی

مثال جالب دیگری که می‌توان ملاحظه نمود. مثال ترسیم نمودار تابع چگالی احتمال نرمال دو بعدی است که فرمول آن نیز در ذیل نمودار آمده است.

```
mu1 <- 0 # setting the expected value of x1
mu2 <- 0 # setting the expected value of x2
s11 <- 10 # setting the variance of x1
s12 <- 15 # setting the covariance between x1 and x2
s22 <- 10 # setting the variance of x2
rho <- 0.5 # setting the correlation coefficient between x1 and x2
x1 <- seq(-10,10,length=41) # generating the vector series x1
x2 <- x1 # copying x1 to x2

f <- function(x1,x2){
    term1 <- 1/(2*pi*sqrt(s11*s22*(1-rho^2)))
    term2 <- -1/(2*(1-rho^2))
    term3 <- (x1-mu1)^2/s11
    term4 <- (x2-mu2)^2/s22
    term5 <- -2*rho*((x1-mu1)*(x2-mu2))/(sqrt(s11)*sqrt(s22))
    term1*exp(term2*(term3+term4-term5)) } # setting up the
# function of the multivariate normal density

z <- outer(x1,x2,f) # calculating the density values

persp(x1, x2, z,
main="Two dimensional Normal Distribution",
```

```
sub=expression(italic(f)~(bold(x))==frac(1,2~pi~sqrt(sigma[11]~
sigma[22]~(1-rho^2)))~phantom(0)~exp~bgroup("",
list(-frac(1,2(1-rho^2)),
bgroup("[", frac((x[1]~-~mu[1])^2,
sigma[11])~-~2~rho~frac(x[1]~-~mu[1],
sqrt(sigma[11]))~ frac(x[2]~-~mu[2],sqrt(sigma[22]))~+~
frac((x[2]~-~mu[2])^2, sigma[22]),"]")),"")),
col="lightgreen",
theta=30, phi=20,
r=50,
d=0.1,
expand=0.5,
ltheta=90, lphi=180,
shade=0.75,
ticktype="detailed",
nticks=5) # produces the 3-D plot
# adding a text line to the graph
mtext(expression(list(mu[1]==0,mu[2]==0,sigma[11]==10,
sigma[22]==10,sigma[12]==15,rho==0.5)), side=3)
```

با اجرای کُدهای فوق نمودار ۲-۳۹ به‌دست می‌آید. البته یک بسته‌ای به نام scatterplot3d برای ترسیم نمودارهای سه بعدی وجود دارد که دارای امکانات بیشتری است. مثال‌هایی از آن در فصل چهارم ارائه خواهد شد.

۲-۲-۵ ذخیره نمودن نمودارها

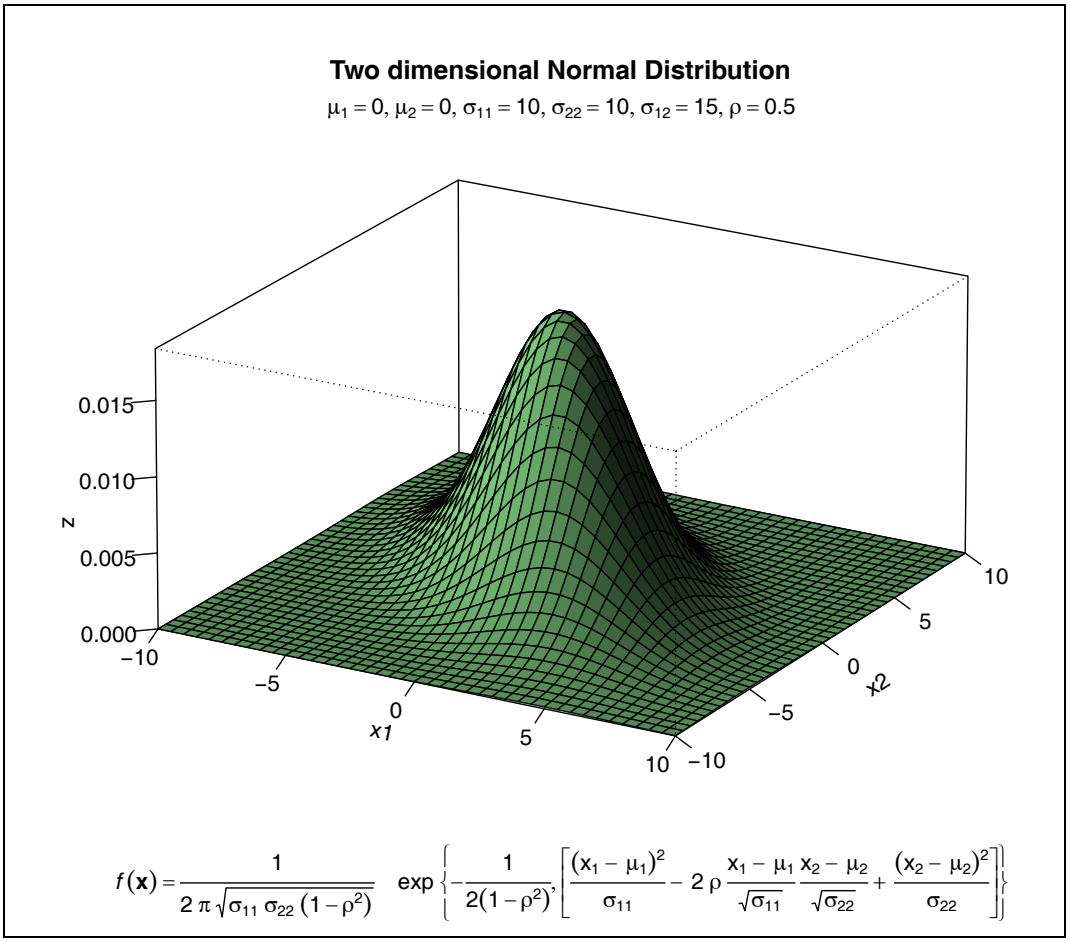
در زبان R می‌توان نمودارهای ترسیم شده را به‌صورت یک فایل در محل مورد نظر و با فرمت‌های متنوع گرافیکی ذخیره نمود. از بین فرمت‌های گوناگون می‌توان از wmf، bmp، png، jpeg، tiff، ps، eps، pdf و ... نام برد.
صورت کلی تابعی که می‌تواند عمل ذخیره را انجام دهد به شرح زیر است.

```
savePlot(file="filename",type="formatname",device=dev.cur())
```

همان‌طور که مشاهده می‌شود با آرگومان type می‌توان فرمت‌های گوناگونی را با اجرای دستور فوق ذخیره کرد. آرگومان device آرگومانی است که شماره پنجره‌ای که قرار است نمودار یا نمودارهای آن ذخیره شود را، به خود اختصاص می‌دهد. اکنون به مثال زیر توجه کنید.

```
> plot(1:10, seq(0,20,l=10))
> savePlot(file="Rplot", type="pdf")
```

با اجرای کُدهای فوق نمودار ناشی از دستور plot به نام Rplot.pdf ذخیره می‌گردد.



شکل ۲-۳۹: نمایش نمودار توزیع نرمال دو متغیره

۶-۲-۲ ذخیره نمودن نمودارها

در زبان R می‌توان نمودارهای ترسیم شده را به‌صورت یک فایل در محل مورد نظر و با فرمت‌های متنوع گرافیکی ذخیره نمود. از بین فرمت‌های گوناگون می‌توان از wmf، bmp، png، jpeg، tiff، ps، eps، pdf و ... نام برد.
صورت کلی تابعی که می‌تواند عمل ذخیره را انجام دهد به شرح زیر است.

```
savePlot(file="filename", type="formatname", device=dev.cur())
```

همان‌طور که مشاهده می‌شود با آرگومان type می‌توان فرمت‌های گوناگونی را با اجرای دستور فوق ذخیره کرد. آرگومان device آرگومانی است که شماره پنجره‌ای که قرار است نمودار یا نمودارهای آن ذخیره شود را، به خود اختصاص می‌دهد. اکنون به مثال زیر توجه کنید.

```
> plot(1:10, seq(0,20,l=10))
> savePlot(file="Rplot", type="pdf")
```

با اجرای کُدهای فوق نمودار ناشی از دستور plot به نام Rplot.pdf ذخیره می‌گردد.

۳-۲ نمودارهای Trellis

همان‌طور که در ابتدای فصل اول آمد زبان R نزدیکی بسیاری با زبان S دارد. در زبان اخیر نمودارهایی را می‌توان ترسیم نمود که Trellis نامیده می‌شود. که می‌تواند نمودارهای سطح بالاتری نسبت نمودارهای پایه ترسیم نماید. در R نیز ترسیم این نمودارها به کمک بسته نرم‌افزاری lattice امکان‌پذیر است. این نرم‌افزار توسط آقای Deepayan Sarkar در دانشگاه Wisconsin آمریکا بسط داده شده است. بسته نرم‌افزاری بالا دارای ویژگی‌های زیر است.

- گرامر دستورات شبیه نمودارهای پایه در R می‌باشد.
- نمودارها را در چند ناحیه ترسیم می‌کند.
- توانایی ایجاد چند سیستم مختصات را دارد.
- قابلیت جانمایی چند لایه در آن موجود است.

شکل کلی پایه یک در تابع در lattice به‌شرح زیر است.

```
vertical.axis.variable ~ horizontal.axis.variable
```

توجه داشته باشید که استفاده از عملگر «~»[8] حتی اگر از یک متغیر هم استفاده شود ضروری است. برای مثال:

```
histogram(~data$x)
xyplot(data$y~data$x)
```

اگر چند نمودار مد نظر باشد، شکل مورد نظر اندکی پیچیده‌تر است. برای مثال:

8. tilde

histogram(~ data$x|data$z)

علامت | مفهوم شرط را بیان می‌کند. متغیر و عبارات طرف راست علامت شرط را متغیرهای شرطی گویند و متغیرهای طرف چپ علامت شرط را متغیرهای اولیه نامند.

در یک فرمول Trellis حداقل یک متغیر اولیه ضرورت دارد، اما متغیر(های) شرطی اختیاری است. اگر متغیر(های) اختیاری موجود نبود علامت شرط باید حذف گردد.

متغیرهای شرطی با علائم * و یا + از هم جدا می‌شوند. اگر متغیرهای شرطی موجود نباشد توابع histogram(~x) و hist(x) و همچنین xyplot(y~x) و plot(x,y) و یا plot(y~x) یکسان عمل می‌کنند.

۲-۳-۱ جمع‌بندی گرامر نمودارهای Trellis

در اینجا پاره‌ای از گرامر مربوط به این گونه نمودارها جمع‌بندی می‌گردد.

- اگر دستور ترسیم به‌صورت vertical~horizontal باشد، فقط یک نمودار ایجاد می‌گردد.
- اگر دستور ترسیم به‌صورت

vertical~horizontal|conditioning variable

باشد، آنگاه برای هر سطح از متغیر شرطی، یک ناحیه ایجاد می‌گردد و در هر ناحیه یک نمودار ترسیم می‌شود.

- اگر دستور ترسیم به‌صورت

vertical~horizontal, group = grouping variable

باشد، آنگاه یک ناحیه ایجاد می‌گردد و نمودارها به‌صورت گروهی در آن ناحیه ترسیم می‌شوند.

- اگر دستور ترسیم به‌صورت

vertical~horizontal|conditioning variable, group = grouping variable

باشد، آنگاه برای هر سطح از متغیر شرطی، یک ناحیه ایجاد می‌گردد و در هر ناحیه نمودارها به‌صورت گروهی در ناحیه مربوط به خودشان ترسیم می‌شوند.

۲-۳-۱-۱ نمودارها با متغیرهایی روی محور طول‌ها

در اینجا منظور نمودارهایی هستند که روی محور طول‌ها متمرکز هستند. به‌عنوان مثال می‌توان از هیستوگرام، چگالی و . . . نام برد. اکنون به حالات گوناگون آن در ادامه توجه کنید.

- اگر دستور ترسیم به‌صورت horizontal~ باشد، فقط یک نمودار ایجاد می‌گردد.
- اگر دستور ترسیم به‌صورت

~horizontal|conditioning variable

باشد، آنگاه برای هر سطح از متغیر شرطی، یک ناحیه ایجاد می‌گردد و در هر ناحیه یک نمودار ترسیم می‌شود.

- اگر دستور ترسیم به‌صورت

~horizontal, group = grouping variable

باشد، آنگاه یک ناحیه ایجاد می‌گردد و نمودارها به‌صورت گروهی در آن ناحیه ترسیم می‌شوند.

- اگر دستور ترسیم به‌صورت

~horizontal|conditioning variable, group = grouping variable

باشد، آنگاه برای هر سطح از متغیر شرطی، یک ناحیه ایجاد می‌گردد و در هر ناحیه نمودارها به‌صورت گروهی در ناحیه مربوط به خودشان ترسیم می‌شوند.

۲-۳-۲ پارامترها

۲-۳-۲-۱ انواع

معمولاً بسته lattice داده‌ها را به‌صورت نقطه در نمودارها ارائه می‌کند. اما آرگومان type می‌تواند این پیش‌فرض را عوض نماید. پاره‌ای از انواع دیگر به‌صورت زیر است.

- type='p': نقاط
- type='l': خطوط
- type='b': نقاط و خطوط
- type='g': شبکه
- type='r': خط رگرسیون
- type='smooth': خط برازش هموار شده
-

۲-۳-۲-۲ محورها

معمولاً بسته lattice برای محورهای نواحی گوناگون یک مقیاس را اختیار می‌کند. برای تغیر از پارامتر scales استفاده می‌شود.

اکنون پس از توضیحاتی راجع به گرامر این نوع از نمودارها، به مثال‌های زیر توجه کنید. البته در ضمن آنها ممکن است نکاتی وجود داشته باشد که همان جا به شرح آن پرداخته خواهد شد.

در بسته MASS داده‌ای به نام whiteside وجود دارد که شامل ستون‌های ایزولاسیون، دمای هوا و مصرف گاز است. به‌عنوان نمونه به چندتای اول و چندتای آخر این داده‌ها توجه کنید.

```
> library(lattice)
> library(MASS)
> data(whiteside)
> head(whiteside)
```

که با اجرای کُدهای بالا نتیجه زیر حاصل می‌شود.

```
 Insul Temp Gas
1 Before -0.8 7.2
2 Before -0.7 6.9
3 Before  0.4 6.4
4 Before  2.5 6.0
5 Before  2.9 5.8
6 Before  3.2 5.8
```

برای انتهای داده‌ها

```
> tail(whiteside)
```

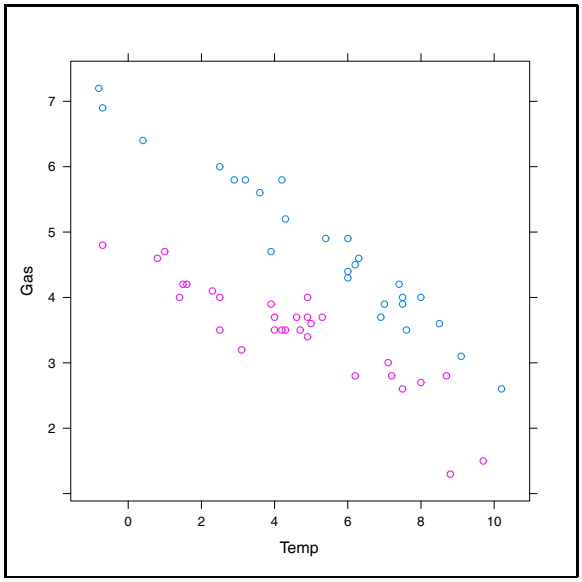
```
 Insul Temp Gas
51 After  7.2 2.8
52 After  7.5 2.6
53 After  8.0 2.7
54 After  8.7 2.8
55 After  8.8 1.3
56 After  9.7 1.5
```

کلمات Before و After به معنای قبل و بعد از ایزولاسیون است. اکنون به اولین مثال این داده‌ها توجه کنید.

```
> library(lattice)
> library(MASS)
> data(whiteside)
> xyplot(Gas ~ Temp, whiteside, group = Insul)
```

که با اجرای کُدهای بالا شکل ۲-۴۰ حاصل می‌شود. شما در شکل ۲-۴۰ تفاوت معنی‌داری را نسبت به نمودارهای پایه R ملاحظه نمی‌کنید، زیرا شرطی وجود ندارد. اما در نمودارهای بعدی تفاوت آشکار می‌گردد.

```
> library(lattice)
> library(MASS)
> data(whiteside)
```



شکل ۲-۴۰: نمایش یک نمودار با دو سری داده

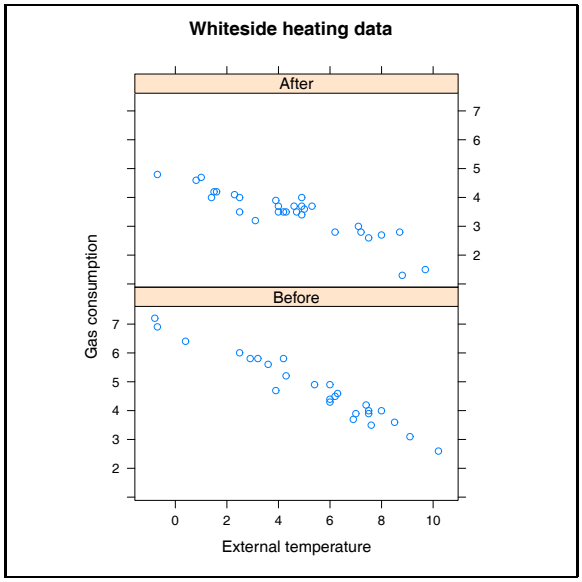
```
> xyplot(Gas ~ Temp | Insul, whiteside,
+ xlab = "External temperature",
+ ylab = "Gas consumption",
+ main = "Whiteside heating data", aspect = 0.6)
```

که با اجرای کُدهای بالا شکل ۲-۴۱ حاصل می‌شود، که نمودارهای پراکنش را در یک صفحه رسم می‌نماید. اکنون به شکل‌های پراکنش توجه کنید که خط‌های رگرسیون نیز به آنها اضافه می‌شود.

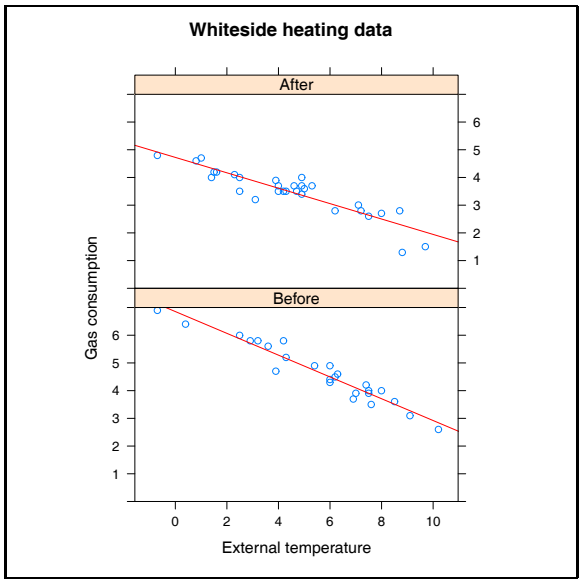
```
> library(lattice)
> library(MASS)
> data(whiteside)
> xyplot(Gas ~ Temp | Insul, whiteside,
+ xlab = "External temperature",
+ ylab = "Gas consumption",
+ main = "Whiteside heating data", aspect = 0.6,
+ panel = function(x, y, ...) {
+ panel.xyplot(x, y, ...)
+ panel.lmline(x, y, ..., col="red")
+ }, ylim = c(0, 7))
```

که با اجرای کُدهای بالا نتیجه شکل ۲-۴۲ حاصل می‌شود.

```
> library(lattice)
> data(iris)
> xyplot(
+ Petal.Length ~ Petal.Width, data = iris, groups=Species,
+ type = c("p", "smooth"), span=.75,
+ auto.key = list(x = 0.15, y = 0.85)
```



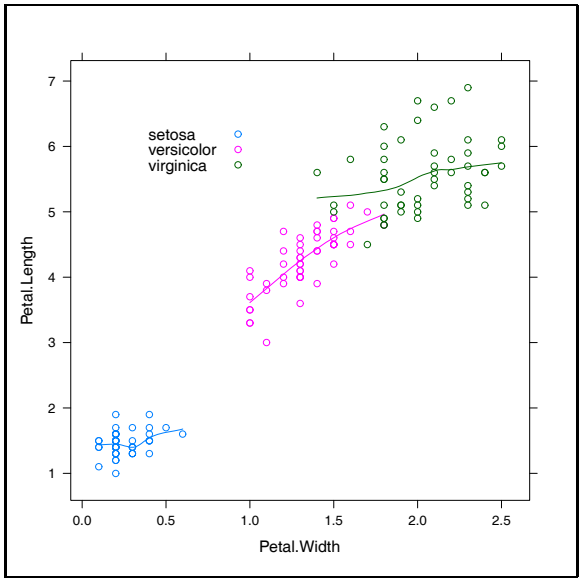
شکل ۲-۴۱: نمایش دو سری داده با استفاده از Trellis



شکل ۲-۴۲: نمایش دو سری داده و خط رگرسیون با استفاده از Trelli

+)

با اجرای کُدهای بالا شکل ۲-۴۳ حاصل می‌شود، که نمودارهای پراکنش را در یک صفحه رسم می‌نماید و خط‌های هموار شده بر آنها برازش یافته است.



شکل ۲-۴۳: نمایش سه سری داده و خط هموار شده با استفاده از Trelli

آرگومان auto.key راهنمای نمودار شکل ۲-۴۳ را ظاهر می‌سازد و برای تعیین موقعیت راهنمای روی نمودار از دستور list استفاده شده است. توجه داشته باشید که مقادیر اعداد داخل دستور اخیر نسبی است و بنابراین در محدود [0,1] قرار می‌گیرد.

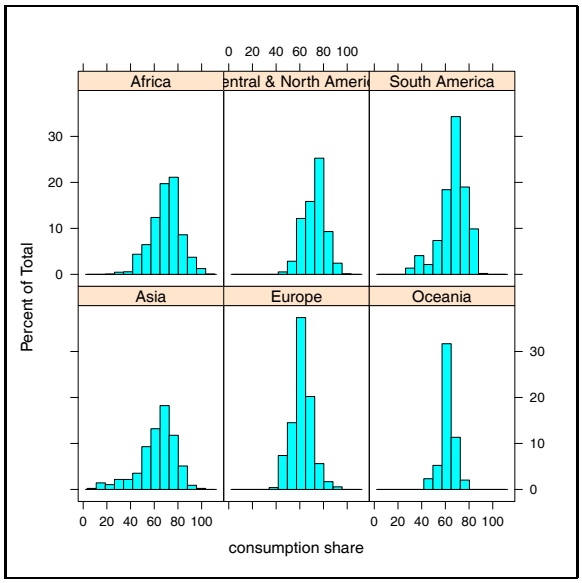
در کُدهای زیر هیستوگرام در ناحیه‌های مختلف ترسیم می‌گردد.

```
> library(pwt)
> library(lattice)
> data(pwt5.6)
> plot(histogram(~c | continent, data = pwt5.6,
+ xlab = "consumption share"))
```

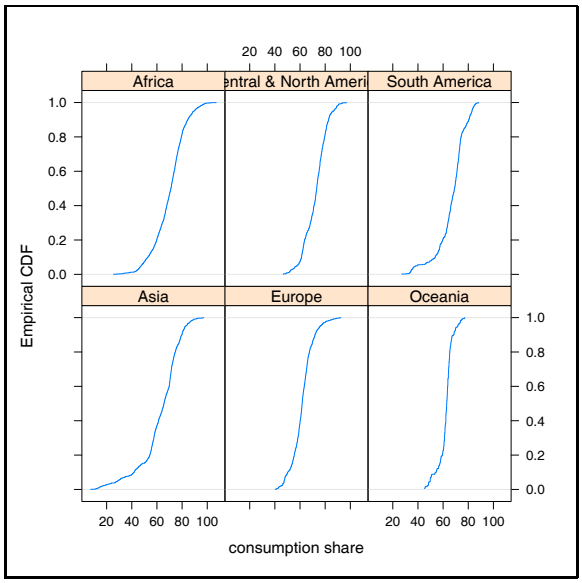
با اجرای کُدهای بالا شکل ۲-۴۴ حاصل می‌شود. در کُدهای زیر منحنی تابع توزیع تجمعی در ناحیه‌های مختلف ترسیم می‌گردد. در اینجا لازم است که از کتابخانه latticeExtra استفاده شود و آن نیز خود به کتابخانه RColorBrewer نیازمند است.

```
> library(pwt)
> library(lattice)
> library(latticeExtra)
> library(RColorBrewer)
> plot(ecdfplot(~c | continent, data = pwt5.6, xlab = "consumption share"))
```

با اجرای کُدهای بالا شکل ۲-۴۵ حاصل می‌شود. در کُدهای زیر منحنی تابع توزیع تجمعی در یک ناحیه ترسیم



شکل ۲-۴۴: نمایش هیستوگرام با استفاده از Trelli

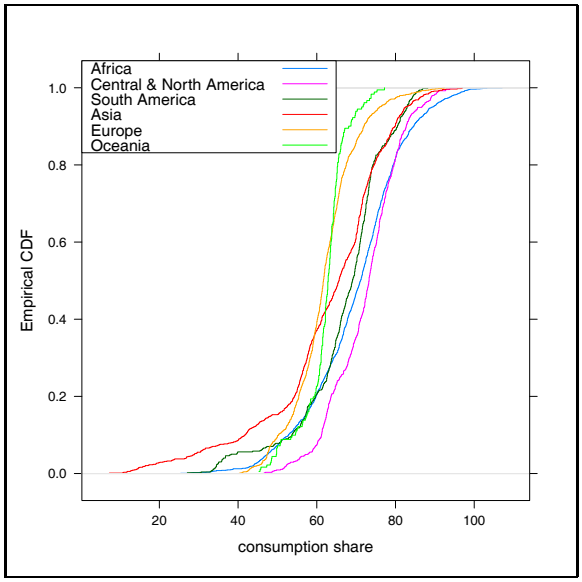


شکل ۲-۴۵: نمایش تابع توزیع تجمعی در چند ناحیه با استفاده از Trelli

می‌گردد.

```
> library(pwt)
> library(lattice)
> library(latticeExtra)
> library(RColorBrewer)
> plot(ecdfplot(~c, groups = continent, data = pwt5.6,
+ auto.key = list(x = 0, y = 1, corner = c(0, 1), background = "white",
+ border = TRUE), xlab = "consumption share"))
```

با اجرای کُدهای بالا شکل ۲-۴۶ حاصل می‌شود.



شکل ۲-۴۶: نمایش تابع توزیع تجمعی در یک ناحیه با استفاده از Trelli

فصل سوم

آمار و احتمال

زبان R شامل تعداد زیادی تابع برای محاسبات آماری، تحلیل داده‌ها و مدل‌سازی آماری است. علاوه بر آن‌ها تعداد قابل ملاحظه‌ای تابع در packageها وجود دارد. در اینجا روی تعداد محدودی از آن‌ها بحث خواهد شد.

۳-۱ توابع پایه آماری

۳-۱-۱ توابع با کاربری زیاد

فهرست توابع در زبان R بسیار زیاد است. فهرست زیر دارای کابرد فراوان می‌باشد.

sum(x): مجموع عناصر x را می‌دهد.

cumsum(x): جمع تجمعی عناصر x را می‌دهد.

prod(x): حاصل‌ضرب عناصر x را می‌دهد.

max(x): حداکثر عناصر x را می‌دهد.

min(x): حداقل عناصر x را می‌دهد.

which.max(x): اندیس بزرگترین عنصر x را می‌دهد.

which.min(x): اندیس کوچکترین عنصر x را می‌دهد.

range(x): این تابع معادل c(min(x), max(x)) است.

length(x): تعداد عناصر x را می‌دهد.

mean(x): میانگین عناصر x را می‌دهد.

median(x): میانه عناصر x را می‌دهد.

var(x): واریانس عناصر x را می‌دهد.

sd(x): انحراف معیار عناصر x را می‌دهد.

cor(x): ماتریس همبستگی را محاسبه می‌کند، اگر x یک ماتریس باشد.

cor(x,y): ضریب همبستگی خطی را بین x و y بدست می‌دهد.

chisq.test(x): آزمون نکویی برازش به روش خی دو است.

ks.test(x): آزمون نکویی برازش به روش کلموگروف-اسمیرنف است.

t.test(x): آزمون tاستودنت برای یک یا دو نمونه است.

var.test(x,y): آزمون برابری واریانس x و y است.

در هسته[1] R محاسبه شاخص‌هایی مانند ضریب چولگی[2] و ضریب کشیدگی[3] وجود ندارد. در زبان R بسته نرم‌افزاری وجود دارد که قادر است نکات گفته شده را عملی سازد. نام این بسته e1071 است. برای این که از مشخصات و فهرست توابع آن آگاه شوید کافی است که دستور زیر را تایپ نموده و کلید Enter را بزنید.

```
> library(help=e1071)
```

از بین توابع حاضر در این بسته دو تابع مورد نظر به ترتیب حروف الفبایی kurtosis و skewness می‌باشد. اکنون به محاسبه ضریب چولگی و ضریب کشیدگی پرداخته می‌شود.

```
> library(class)
> library(e1071)
> x <- trees$Height
> skewness(x)
[1] -0.3568773
> kurtosis(x) [1] -0.7233677
```

لازم به توضیح است که برای استفاده از بسته e1071 باید بسته class نیز حاضر باشد. داده مورد استفاده نیز یکی از مؤلفه‌های داده trees است که در خود R موجود است.

1. core 2. skewness coefficient 3. kurtosis coefficient

۳-۱-۲ توزیع‌های احتمالی و اعداد تصادفی

بیشتر توابع احتمالی در زبان R گذاشته شده است. هر تابع دارای چهار شکل متفاوت است، که به صورت زیر هستند.

- دستور dfunc(x, ⋯)، عرض تابع را در نقطه x نشان می‌دهد.
- دستور pfunc(x, ⋯)، مقدار احتمال تجمعی را تا نقطه x نشان می‌دهد.
- دستور qfunc(p, ⋯) مقدار چندک تابع را به ازای $0 < p < 1$ نشان می‌دهد.
- دستور rfunc(x, ⋯)، نمونه تصادفی از تابع را شبیه‌سازی می‌کند.

در جدول ۳-۱ فهرست یک سری از توابع احتمالی را نشان می‌دهد، که پارامترها به صورت پیش‌فرض است و کاربر می‌تواند آن‌ها را حسب مورد سفارشی نماید.

به مثال زیر در مورد عدد تصادفی که دارای تابع احتمال نرمال است توجه کنید.

```
> rnorm(1)
[1] 1.358007
```

هر بار که تابع مورد نظر اعداد تصادفی را اجرا می‌کنید یک سری جدیدی از اعداد تصادفی ایجاد می‌گردد، که با سری پیشین متفاوت است. اکنون اگر از دستور set.seed() قبل از اجرای تابع اعداد تصادفی استفاده کنید، آنگاه در هر اجرا همان سری را خواهید یافت که قبلاً تولید شده بود. آرگومان تابع اخیر یک عدد صحیح دلخواه است.

برای بدست آوردن احتمال تجمعی $F_X(x) = \Pr(X \leq x)$ یک توزیع به مثال زیر توجه کنید.

```
> pnorm(1.96)
[1] 0.9750021
```

اکنون می‌توان با یک امکان اضافی احتمال $1 - F_X(x) = \Pr(X > x)$ را نیز بدست آورد.

```
> pnorm(1.96, lower.tail=F)
[1] 0.02499790
```

برای بدست آوردن مقادیر بحرانی و یا P-value در آزمون آماری مورد استفاده قرار می‌گیرد. مثلاً برای مقادیر بحرانی آزمون دو طرفه توزیع نرمال برای ۵٪ به صورت زیر است.

```
> qnorm(0.025)
[1] -1.959964
> qnorm(0.975)
[1] 1.959964
```

مقدار P-value برای آزمون $\chi^2 = 3.84$ با df=1 برابر است با:

```
> 1-pchisq(3.84, 1)
[1] 0.05004352
```

به یک مثال دیگر توجه کنید. مقدار P-value عدد 3.6 را در تابع f(4,43) حساب کنید.

جدول ۳-۱: توزیع‌های آماری و توابع آن

توزیع	کُد	پارامترها	پیش‌فرض	package
beta	beta	shape1, shape2	-, -	stat*
binomial	binom	size, prob	-, -	stat
Cauchy	chauchy	location, scale	0, 1	stat
χ^2	chisq	df, ncp	-, 0	stat
Dirichlet	dirichlet	alpha	-	MCMCpack
exponential	exp	rate	1	stat
F	f	df1, df2, ncp	-, -, -	stat
gamma	gamma	shape, rate, scale	-, 1, 1/rate	stat
geometric	geom	prob	-	stat
Generalized Extreme Value	gev	xi,mu,sigma	-, -, -	evir
Generalized Pareto	gpd	xi,mu,beta	-, -, -	evir,POT
hypergeometric	hyper	m, n, k	-, -, -	stat
Inverse Gamma	invgamma	shape,rate	-, -	MCMCpack
Inverse Wishart	iwish	v, S	-, -	MCMCpack
logistic	logis	location, scale	0, 1	stat
lognormal	lnorm	meanlog, sdlog	0, 1	stat
Multinomial	multinom	size, prob	-, -	stat
lognormal	lnorm	meanlog, sdlog	0, 1	stat
Multivariate Normal	mvnorm	mean,sigma	-, -	mvtnorm
Multivariate-t	mvt	sigma,df	-, -	mvtnorm
negative binomial	nbinom	size, prob, mu	-, -, -	stat
nomal	norm	mean, sd	0,1	stat
Poisson	pois	lambda	-	stat
'Student' (t)	t	df, ncp	-, 0	stat
Weibull	weibull	shape, scale	-, 1	stat
uniform	unif	min, max	0,1	stat
Wilcoxon	wilcoxon	m, n	-, -	stat
Wishart	wish	v, S	-, -	MCMCpack

*) منظور پایه و یا هسته اصلی خود زبان R است.

```
> 1-pf(3.6,4,43)
[1] 0.01284459
```

برای سطح معنی‌دار بودن 1% رد نمی‌شود زیرا بزرگتر از آن است. اما برای سطح معنی‌دار بودن 5% رد می‌گردد چون کوچکتر از آن است. فراموش نشود که اگر آزمون دو طرفه بود P-value باید در دو ضرب شود.

```
> 1-pt(2.8,21)
[1] 0.005364828
> 2*(1-pt(2.8,21))
[1] 0.01072966
```

اکنون اگر سطح معنی‌دار بودن را داشته باشید می‌توانید چندک‌های آن را محاسبه کنید.

```
> alpha <- c(0.1, 0.05, 0.01, 0.001)
> qnorm(1-alpha/2)
[1] 1.644854 1.959964 2.575829 3.290527
```

۳-۱-۲-۱ هیستوگرام

در آمار و احتمال هیستوگرام، نمودار معروفی است. که صورت کلی آن به شرح زیر است:

```
hist(x,breaks="Sturges",prob=FALSE)
```

آرگومان breaks تعداد دسته‌ها را نشان می‌دهد. تعداد دسته‌ها خیلی زیاد یا خیلی کم می‌تواند شکل نامناسبی را ایجاد کند. در صورت پیش فرض زبان R از فرمول Sturges استفاده می‌کند که رابطه آن به شکل زیر است.

$$\lceil \log_2(n) + 1 \rceil$$

روش‌های دیگری نیز وجود دارد که پهنای دسته را معین می‌کند، مثل فرمول Freedman-Diaconis (FD) که بر اساس محدوده بین چارکی (iqr) قرار دارد. رابطه آن به شکل زیر است.

$$2 \times \text{iqr} \times n^{-\frac{1}{3}}$$

فرمول دیگری نیز وجود دارد که مبنای آن فرمول زیر است و توسط Scott پیشنهاد شده است.

$$3.5 \times s \times n^{-\frac{1}{3}}$$

که در آن s انحراف معیار است.

اگر بخواهید کنترل کاملاً در اختیار شما باشد می‌توانید از آرگومان nclass=num. of class استفاده کنید. به‌عنوان مثال nclass=20 برای هیستوگرام مورد نظر 20 دسته ایجاد می‌کند.

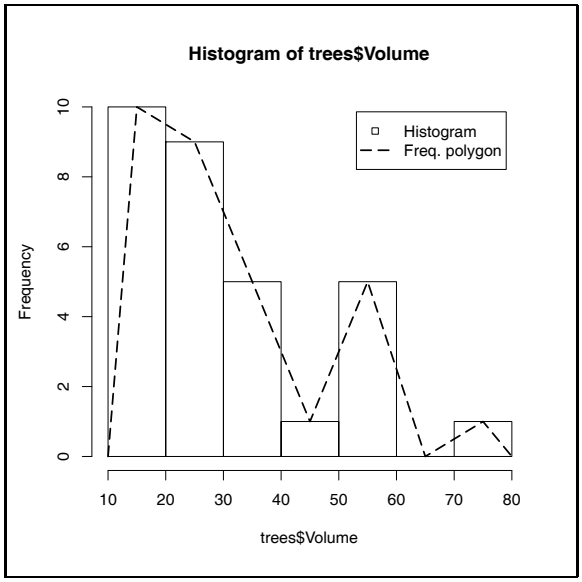
در دستور freq=F و یا prob=T فراوانی غیرفعال می‌شود و در واقع چگالی ظاهر می‌گردد و شکل به‌گونه‌ای تنظیم می‌گردد که مجموع مساحت‌ها در هیستوگرام برابر واحد شود.

چندضلعی فراوانی: همراه با نمودار هیستوگرام نموداری دیگری را می‌توان رسم نمود که چندضلعی فراوانی

نامیده می‌شود و از شهرت به‌سزایی در آمار توصیفی برخوردار است. برای ترسیم نمودار گفته شده به مثال زیر توجه کنید.

```
> tmp <- hist(trees$Volume)
> lines(c(min(tmp$breaks),tmp$mids,max(tmp$breaks)),c(0,tmp$counts,0),
+ lty=5,lwd=1.75)
> legend("topright",c("Histogram","Freq. polygon"),lty=c(NA,5),lwd=c(NA,1.75),
+ pch=c(22,NA),inset = 0.05)
```

در کُدهای فوق ابتدا تمام مشخصات هیستوگرام در شی به نام tmp قرار می‌گیرد. سپس در تابع lines() از آرگومان‌های هیستوگرام استفاده می‌شود. اولین آرگومان breaks است که دسته‌ها را نشان می‌دهد. دومین آرگومان mids است که بردار نقاط وسط دسته‌ها را نشان می‌دهد. سومین آرگومان counts است که تعداد فراوانی هر دسته نشان می‌دهد.
با اجرای کُدهای فوق شکل ۳-۱ حاصل می‌گردد.

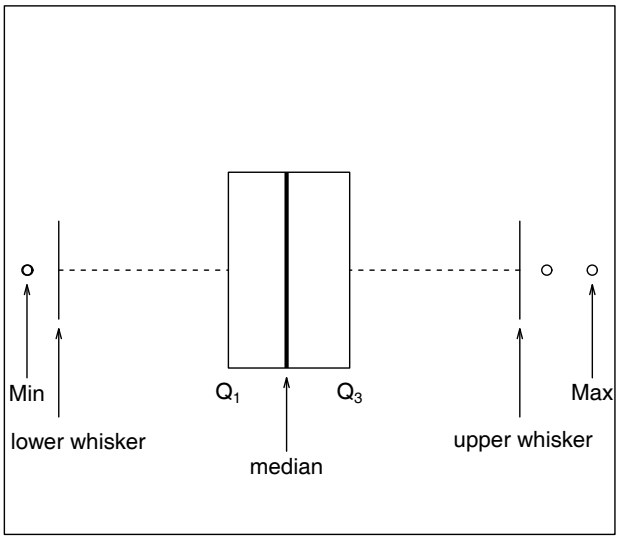


شکل ۳-۱: نمایش نمودارهای هیستوگرام و چندضلعی فراوانی

۳-۱-۲-۲ باکس پلات

یکی از نمودارهای مشهور در آمار boxplot است. این نمودار به‌خوبی توزیع داده‌ها حول میانه را نشان می‌دهد و همچنین داده‌های چوله و داده‌های پرت را به‌دست می‌دهد. برای توضیح عناصر این نمودار ابتداً به شکل ۳-۲ توجه کنید. که در آن Q_1 و Q_3 به ترتیب چارک اول و چارک سوم است. نقاطی که بیرون از دو خط عمودی در طرفین باکس قرار دارند به نقاط پرت[4] مشهورند. همانطور که ملاحظه می‌کنید boxplot() دارای آرگومان منطقی horizontal است که در حالت پیش‌فرض نادرست «FALSE» است و شکل باکس پلات از حالت

4. outliers



شکل ۳-۲: نمایش قسمت‌های مختلف boxplot

افقی به حالت عمودی در می‌آید.

اکنون به مثالی توجه کنید که دارای چند گروه داده باشد. برای انجام این کار از داده trees که جزء داده‌های R است استفاده می‌گردد. حاصل شکل ۳-۳ است.

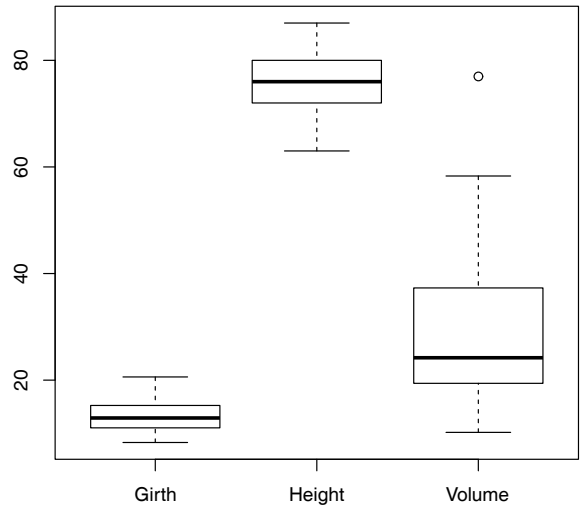
```
> boxplot(trees)
```

اما چگونه می‌توان مشخصات عددی boxplot را به‌دست آورد. در تابع گرافیکی آرگومان منطقی plot وجود دارد که مقدار پیش‌فرض آن درست «TRUE» است. اگر مقدار این آرگومان نادرست شود، آنگاه مقادیر عددی ظاهر می‌شود.

```
> boxplot(trees, plot=FALSE)
```

حاصل مقادیر زیر است:

```
$stats
 [,1] [,2] [,3]
[1,]  8.30   63 10.2
[2,] 11.05   72 19.4
[3,] 12.90   76 24.2
[4,] 15.25   80 37.3
[5,] 20.60   87 58.3
```



شکل ۳-۳: نمایش boxplot داده‌های trees

```
$n
[1] 31 31 31

$conf
 [,1]     [,2]    [,3]
[1,] 11.70814 73.72979 19.1204
[2,] 14.09186 78.27021 29.2796

$out
[1] 77

$group
[1] 3

$names
[1] "Girth"  "Height" "Volume"
```

همان‌طور که ملاحظه می‌شود مقادیر فوق به‌صورت یک لیست هستند که مؤلفه‌های آن به شرح زیر است.

stats: ماتریسی است که پنج عنصر هر گروه که شامل whisker پایین، چارک اول، میانه، چارک سوم و whisker بالا است را نشان می‌دهد.

n: برداری است که تعداد عناصر هر گروه را معین می‌کند.

conf: ماتریسی است که مقادیر پایین و بالا notch هر گروه را نشان می‌دهد.

out: مقادیر خارج از whiskerها را نشان می‌دهد که در واقع داده‌های پرت هستند.

group: تعداد گروه‌های یک داده را نشان می‌دهد.

names: نام هر گروه یک داده‌ها را نشان می‌دهد.

۳-۲-۱-۳ تابع table()

برای خلاصه نمودن داده‌ها می‌توان از تابع table() استفاده نمود. برای مثال در R مجموعه داده‌ای تحت عنوان mtcars وجود دارد که 11 خصوصیت از 32 خودرو را شامل می‌شود. قبلاً نیز در فصل اول از آن استفاده شده است. اکنون کُدهای زیر برای خصوصیت دوم مجموعه داده‌ی فوق یعنی cyl (سیلندر) اعمال می‌گردد.

```
> data(mtcars)
> attach(mtcars)
> table(cyl)
```

حاصل مقادیر زیر است:

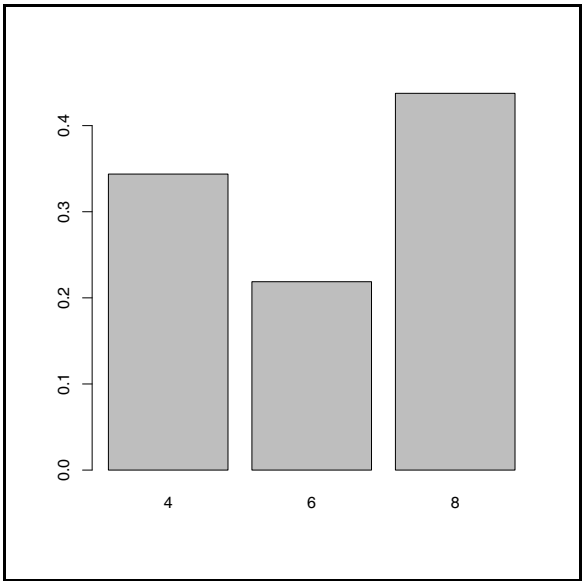
```
cyl
 4  6  8
11  7 14
```

بنابراین ملاحظه می‌شود که 11 خودرو 4 سیلندر، 7 خودرو 6 سیلندر و 14 خودرو 8 سیلندر وجود دارد. اکنون می‌توان خلاصه را به‌صورت فراوانی نسبی نیز ارائه نمود.

```
> table(cyl)/length(cyl)
```

```
cyl
     4      6      8
0.34375 0.21875 0.43750
```

خلاصه را می‌توان به‌صورت نمودار نیز ترسیم نمود. این کار توسط تابعی تحت عنوان barplot() قابل ارائه است. آرگومان تابع شی است که توسط تابع table() ایجاد شده است. شکل ۳-۴ ترسیم خلاصه را نشان می‌دهد.



شکل ۳-۴: نمایش نمودار table

تابع table() مشتقاتی نظیر margin.table() و prop.table() نیز دارد. که اولین تابع حواشی یک جدول را معین می‌کند و دومی فراوانی نسبی جدول مشخص می‌سازد. اکنون به مثال زیر توجه کنید.

```
> m <- matrix(1:6,2)
> m
```

حاصل مقادیر زیر است:

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

اکنون تابع margin.table() روی m اعمال می‌گردد.

```
> margin.table(m,1)
[1] 9 12
> margin.table(m,2)
[1] 3 7 11
```

آرگومان دوم تابع اخیر اگر 1 باشد نماینده سطر و اگر 2 باشد نماینده ستون جدول است. اگر آرگومان دوم منظور نشود، تابع، مجموع کل عناصر جدول را به‌دست می‌دهد.
اکنون تابع prop.table() روی m اعمال می‌گردد.

```
> prop.table(m,1)
```

حاصل مقادیر زیر است:

```
 [,1]      [,2]      [,3]
[1,] 0.1111111 0.3333333 0.5555556
[2,] 0.1666667 0.3333333 0.5000000
```

یعنی عناصر سطر اول جدول بر عدد 9 و عناصر سطر دوم آن بر عدد 12 تقسیم شده است. همان‌طورکه ملاحظه می‌شود مجموع عناصر هر سطر جدول اخیر برابر واحد است.

```
> sum(prop.table(m,1)[1,])
[1] 1
```

اکنون به بُعد دوم جدول توجه کنید.

```
> prop.table(m,2)
```

حاصل مقادیر زیر است:

```
 [,1]    [,2]      [,3]
[1,] 0.3333333 0.4285714 0.4545455
[2,] 0.6666667 0.5714286 0.5454545
```

یعنی عناصر ستون اول جدول بر عدد 3، عناصر ستون دوم آن بر عدد 11 و عناصر ستون سوم بر عدد 11 تقسیم شده است. همان‌طورکه ملاحظه می‌شود مجموع عناصر هر ستون جدول اخیر برابر واحد است.

```
> sum(prop.table(m,2)[,1])
[1] 1
```

برای تابع table() می‌توان مثال زیر را مد نظر قرار داد، که در آن 1500 داده تصادفی از توزیع پواسون با پارامتر 1.5 ایجاد می‌گردد و سپس خلاصه می‌شود.

```
> set.seed(10)
> y <- rpois(1500,1.5)
> table(y)
```

حاصل مقادیر زیر است:

```
y
  0   1   2   3   4   5   6   7   8
336 522 350 193  59  26  11   2   1
```

که در آن تعداد صفرها 336، تعداد یک‌ها 522 و . . . می‌باشد.

اکنون به سادگی می‌توان جدول اخیر را به یک داده چارچوب‌دار با دو متغیر تبدیل نمود. برای این کار از تابع

as.data.frame() استفاده می‌شود.

```
> as.data.frame(table(y))
```

حاصل مقادیر زیر است:

```
  y Freq
1 0  336
2 1  522
3 2  350
4 3  193
5 4   59
6 5   26
7 6   11
8 7    2
9 8    1
```

اکنون به چند تابع احتمال پرداخته می‌شود.

۳-۱-۲-۴ تابع چگالی احتمال نرمال

تابع چگالی احتمال نرمال به‌صورت زیر بیان می‌شود:

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

برای نمایش تابع چگالی احتمال و تابع تجمعی آن به شکل زیر توجه کنید.

```
X <- seq(-4,4,0.1)
par(mfrow=c(1,2))
plot(X,dnorm(X),type='l',xlab="x", ylab="f(x)", main="PDF's Normal")
plot(X,pnorm(X),type='l',xlab="x", ylab="F(x)", main="CDF's Normal")
```

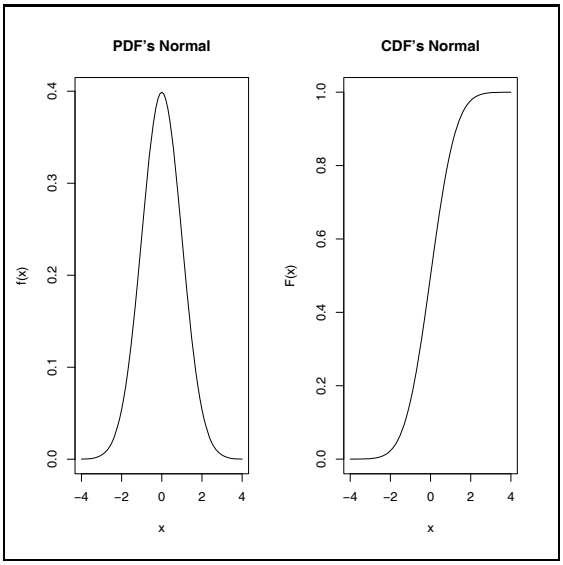
با اجرای کُدهای بالا، شکل ۳-۵ حاصل می‌گردد. اکنون می‌توان با اعداد تصادفی نرمال، تابع مذکور را شبیه‌سازی نمود و آن را با تابع نظری مقایسه کرد (شکل ۳-۶).

```
set.seed(5)
Y <- rnorm(2000) # 2000 from Normal(mu = 0, sigma=1)
```

در دستور زیر اعداد کوچکتر از 4- و بزرگتر از 4 حذف می‌شوند.

```
Y[Y < -4 | Y > 4] <- NA
x <- seq(-4, 4, .1)
```

در دستور زیر فراوانی غیرفعال می‌شود تا بتوان هیستوگرام و تابع چگالی را در یک شکل رسم نمود.



شکل ۳-۵: نمایش نمودارهای تابع چگالی احتمال و تجمعی نرمال

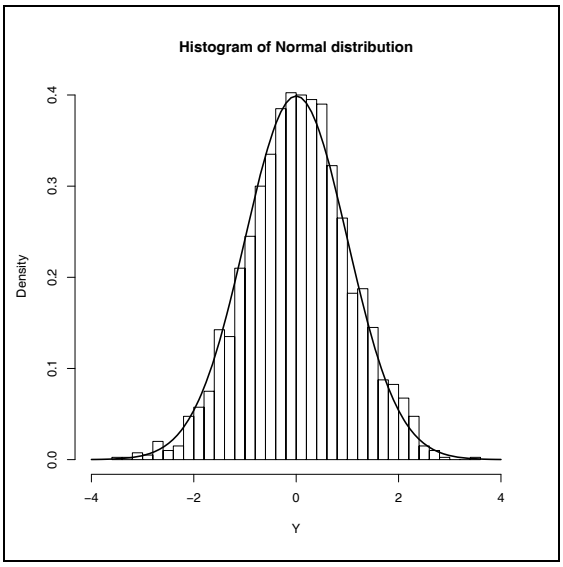
```
hist(Y, breaks="FD", xlim=c(-4,4), freq=FALSE, main="")
title("Histogram of Normal distribution")
lines(x, dnorm(x), lwd=2)
```

۳-۱-۲-۵ توابع گرافیکی qqnorm()، qqline() و qqplot()

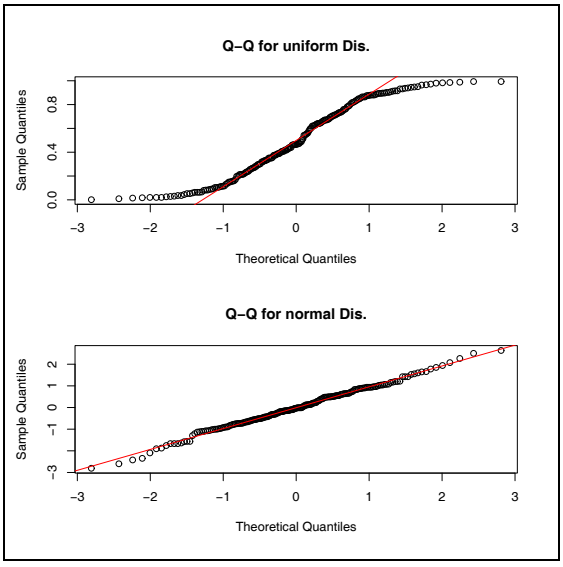
می‌توان نرمال بودن یک سری داده را با تابع qqnorm() نشان داد. اگر داده‌ها نرمال باشند، آنگاه یک خط راست را نمایش می‌دهند. ترسیم خط راست از تابع qqline() حاصل می‌شود که برای مقایسه (شکل ۳-۷) به‌کار می‌رود. به مثال زیر توجه کنید.

```
x <- runif(200)
y <- rnorm(200)
par(mfrow=c(2,1))
qqnorm(x,main="")
title("Q-Q for uniform Dis.")
qqline(x, col="red")
qqnorm(y, main="")
title("Q-Q for normal Dis.")
qqline(y, col="red")
```

اگر دو سری داده داشته باشید، می‌توان این سوال را مطرح نمود که آیا این دو سری از یک توزیع شده‌اند؟ برای انجام این کار در زبان R تابعی به صورت qqplot() وجود دارد. با رسم نمودار اگر نقاط تشکیل یک خط راست را بدهند، آنگاه فرض توزیع مشترک قابل قبول است. برای مقایسه نیز می‌توان از یک خط راست استفاده نمود. این خط نیمساز ربع اول است که با استفاده از تابع abline(0,1) حاصل می‌گردد (شکل ۳-۸).

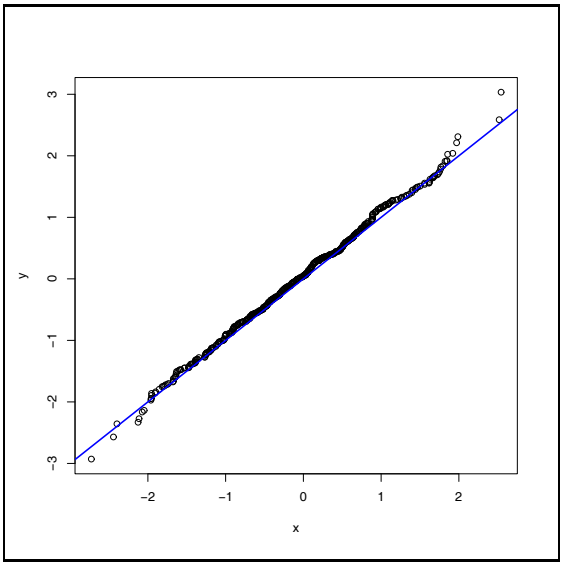


شکل ۳-۶: نمایش نمودارهای هیستوگرام و تابع احتمال نظری



شکل ۳-۷: نمایش نمودارهای چندک‌های نرمال دو تابع چگالی احتمال یکنواخت و نرمال

```
x <- rnorm(500)
y <- rnorm(500)
qqplot(x,y)
abline(0,1, col="blue", lwd=2)
```



شکل ۳-۸: نمایش نمودار چندک‌های دو سری داده

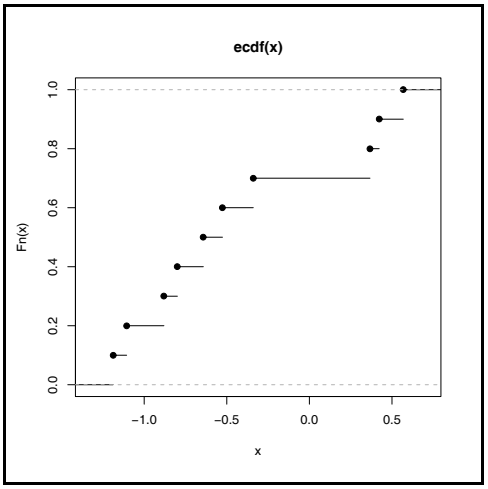
۳-۱-۲-۶ تابع ecdf()

همانطورکه قبلاً ملاحظه شد. می‌توان تابع توزیع تجمعی را محاسبه و ترسیم کرد. اما تابعی وجود دارد که ecdf() نامیده می‌شود و توزیع تجمعی تجربی را بدست می‌دهد. به مثال زیر توجه کنید.

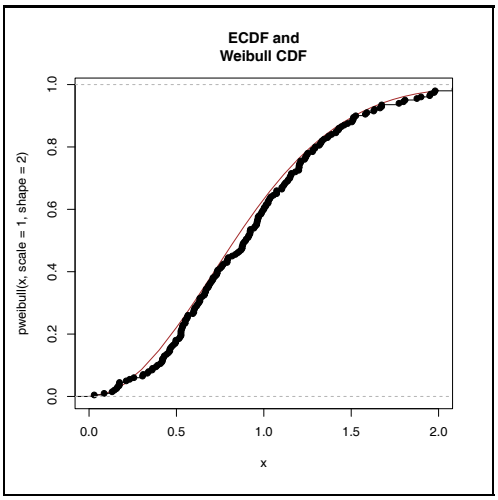
```
> x <- rnorm(10)
> plot(ecdf(x))
```

پس از ترسیم، شکل ۳-۹ به‌دست می‌آید.
اکنون به کُدها و نمودار ۳-۱۰ توجه کنید که توزیع احتمال تجمعی ویبول را (در حالت تجربی و نظری) نشان می‌دهد.

```
> x <- seq(0,2,0.1)
> plot(x, pweibull(x,scale=1,shape=2),type="l", main="ECDF and Weibull CDF")
> x.teo <- rweibull(n=200,shape=2, scale=1)
> plot(ecdf(x.teo), add=TRUE)
```



شکل ۳-۹: نمایش نمودار توزیع تجمعی تجربی نرمال



شکل ۳-۱۰: نمایش نمودار توزیع تجمعی تجربی ویبول

۳-۱-۲-۷ تابع چگالی احتمال گاما

قبل از بیان این تابع چگالی احتمال، ابتدا لازم است که تابع گاما تعریف شود.

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-x}\, dx \qquad \alpha > 0$$

سپس می‌توان تابع چگالی احتمال گاما را به صورت زیر بیان نمود.

$$f(x) = \begin{cases} \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-\frac{x}{\beta}} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

این تابع دارای دو پارامتر α (پارامتر شکل shape) و β (پارامتر مقیاس scale) نامیده می‌شود. اگر مقدار $\alpha = 1$ باشد، این تابع به توزیع نمایی تبدیل می‌شود.
گشتاورهای این تابع چگالی احتمال به صورت زیر است.

$$\mu_X = \alpha\beta$$
$$\sigma_X^2 = \alpha\beta^2$$

از حل معادلات فوق روابط زیر بدست می‌آید.

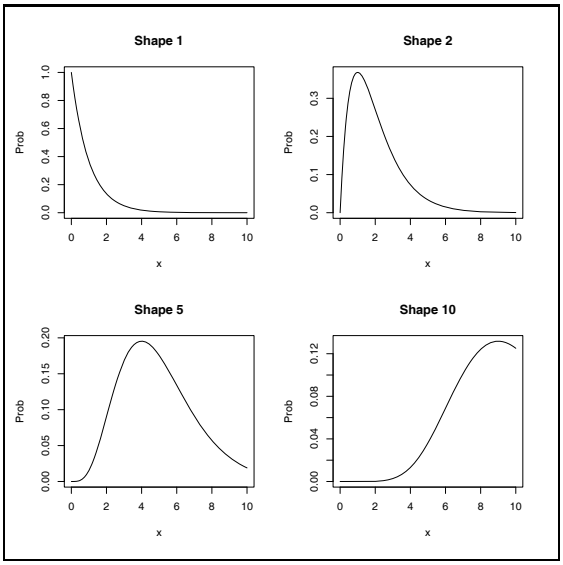
$$\beta = \frac{\sigma_X^2}{\mu_X}$$
$$\alpha = \left(\frac{\mu_X}{\sigma_X}\right)^2$$

اکنون به مثال‌های R در مورد این توزیع (شکل ۳-۱۱) توجه کنید.

```
x <- seq(0,10, length=100)
par(mfrow=c(2,2))
plot(x,dgamma(x,shape=1,scale=1), type='l',xlab="x",
ylab="Prob", main="Shape 1")
plot(x,dgamma(x,shape=2,scale=1), type='l',xlab="x",
ylab="Prob", main="Shape 2")
plot(x,dgamma(x,shape=5,scale=1), type='l',xlab="x",
ylab="Prob", main="Shape 5")
plot(x,dgamma(x,shape=10,scale=1), type='l',xlab="x",
ylab="Prob", main="Shape 10")
```

اکنون به مثال پارامتر scale توجه کنید (شکل ۳-۱۲).

```
x <- seq(0,30,length=100)
plot(x,dgamma(x,shape=2,scale=1), type='l', xlab="x",
```



شکل ۳-۱۱: نمایش نمودارهای توزیع گاما با shapeهای مختلف

```
ylab="f(x)", main="Gamma pdf's")
lines(x,dgamma(x,shape=2,scale=2), lty=2)
lines(x,dgamma(x,shape=2,scale=4), lty=3)
lines(x,dgamma(x,shape=2,scale=8), lty=4)
legend(x=20,y=.35,paste("Scale=",c(1,2,4,8)), lty=1:4)
```

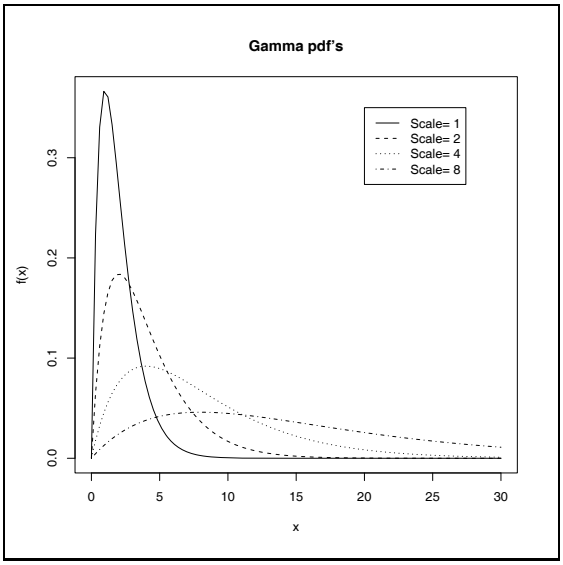
می‌توان بر یک سری داده‌های مورد نظر، توزیع گاما را برازش داد. به مثال زیر توجه کنید.

```
X <- c(4.75, 3.4, 1.8, 2.9, 2.2, 2.4, 5.8, 2.6, 2.4, 5.25)
n <- length(X)
alpha <- (mean(X)/sd(X))^2
theta <- var(X)/mean(X)
x <- seq(0, 8, length=200)
plot(x, dgamma(x,shape=alpha,scale=theta), type='l', ylab="f(x)")
points(X, rep(0,n))
```

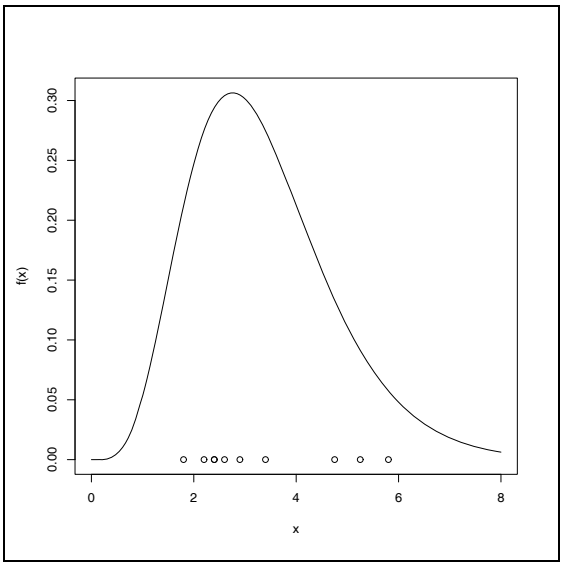
با اجرای کُدهای فوق شکل ۳-۱۳ حاصل می‌شود. البته می‌توان مثال بالا را تکرار نمود با این تفاوت که نحوه‌ی نشان دادن داده‌های تجربی (مثال قبل) به صورت تیک‌های محوری باشد.

```
X <- c(4.75, 3.4, 1.8, 2.9, 2.2, 2.4, 5.8, 2.6, 2.4, 5.25)
n <- length(X)
alpha <- (mean(X)/sd(X))^2
theta <- var(X)/mean(X)
x <- seq(0, 8, length=200)
plot(x, dgamma(x,shape=alpha,scale=theta), type='l', ylab="f(x)")
rug(X, ticksize = 0.03, side = 1, lwd = 2)
```

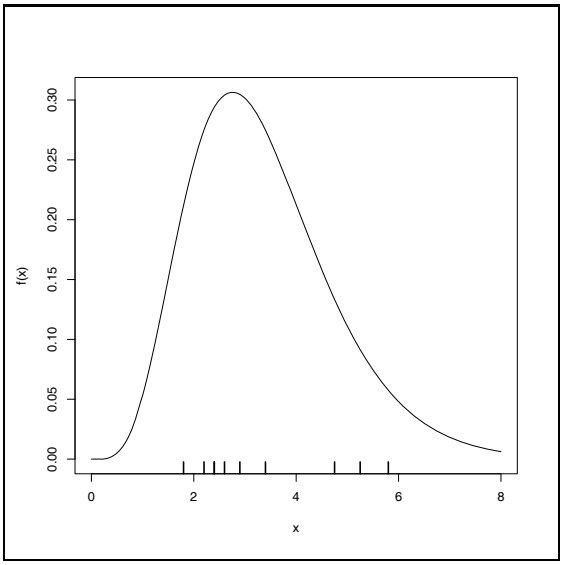
با اجرای کُدهای فوق شکل ۳-۱۴ حاصل می‌شود.



شکل ۳-۱۲: نمایش نمودارهای توزیع گاما با scaleهای مختلف



شکل ۳-۱۳: نمایش برازش تابع گاما بر یک سری داده



شکل ۳-۱۴: نمایش برازش تابع گاما بر یک سری داده (نقاط تجربی به صورت تیک)

۳-۱-۲-۸ تابع چگالی احتمال پیرسن[5]

این خانواده از توابع چگالی احتمال، از پیرسن نوع صفر شروع می‌شود و تا پیرسن نوع 7 ادامه می‌یابد. البته تابع چگالی احتمال پیرسن صفر تقریباً همان تابع چگالی احتمال نرمال است، زیرا ضریب چولگی آن صفر و ضریب کشیدگی آن برابر 3 می‌باشد. در پاره‌ای از مباحث، تابع چگالی احتمال پیرسن نوع 3 کاربرد بیشتری دارد. این نوع پیرسن شبه توزیع گاما می‌باشد که فرمول آن به‌صورت زیر است.

$$f(x) = \frac{1}{|s|^a \Gamma(a)} |x - \lambda|^{a-1} e^{-\frac{x-\lambda}{s}}$$

که در آن $a > 0$ پارامتر شکل، $s \neq 0$ پارامتر مقیاس و λ پارامتر مکان است. ضمناً $\frac{x-\lambda}{s} \geq 0$ می‌باشد. توابع این خانواده از توزیع‌ها در هسته R موجود نیست و باید از بسته PearsonSD استفاده نمود. در این بسته می‌توان چهارگشتاور اول تجربی داده‌ها را به‌صورت زیر محاسبه نمود.

```
> library(PearsonDS)
> x <- c(2,5,9,10,8,11,15,13,18,22,20,32,25,21,26,29,35,45,40,50,60,100,
+ 30,55,58,65,77,66,70,170,100,105,48,61,59,64,65,70,72)
> empMoments(x)
```

حاصل کُد بالا به‌صورت زیر است.

```
     mean      variance  skewness  kurtosis
46.948718  1157.074293  1.236126  5.343204
```

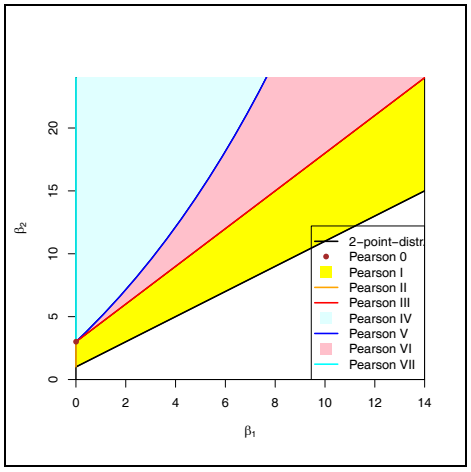
در این بسته می‌توان دیاگرامی برای نواحی مختلف توزیع‌های پیرسن ترسیم نمود. به کُد زیر توجه کنید.

```
> library(PearsonDS)
```

5. Pearson

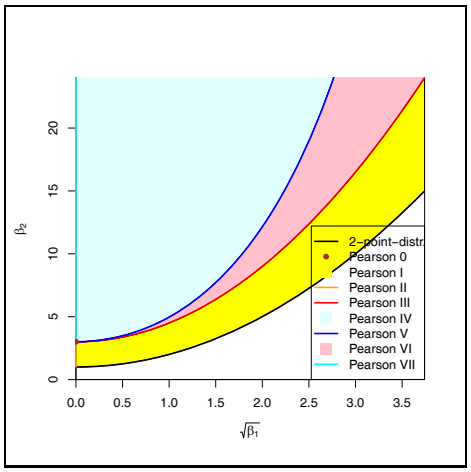
```
> pearsonDiagram(max.skewness = sqrt(14), max.kurtosis = 24,
+ squared.skewness = TRUE, lwd = 2, legend = TRUE,
+ n = 301)
```

حاصل کُد بالا شکل ۳-۱۵ است. در کُد بالا آرگومان squared.skewness = TRUE است. اکنون اگر این



شکل ۳-۱۵: نمایش دیاگرام توزیع‌های پیرسن

آرگومان به‌صورت squared.skewness = FALSE تغییر کند، آنگاه با تغییر محورها طول‌ها شکل ۳-۱۶ حاصل می‌شود. همان‌طور که قبلاً ملاحظه شد برای تابع احتمال چهار نوع وجود داشت که با حروف d، p، q و



شکل ۳-۱۶: نمایش دیاگرام توزیع‌های پیرسن با محور $\sqrt{\beta_1}$

r در ابتدای نام تابع مشخص می‌شد. این قاعده در مورد انواع توزیع‌های پیرسن نیز صادق است. برای مثال به کد توزیع پیرسن نوع سوم توجه کنید.

```
> library(PearsonDS)
```

```
> set.seed(1)
> pIIIpars <- list(shape=3, location=1, scale=-0.5)
> # calculate probability density function
> dpearsonIII(-4:1,params=pIIIpars)
[1] 0.004539993 0.021469608 0.089235078 0.293050222 0.541341133 0.000000000
> # calculate cumulative distribution function
> ppearsonIII(-4:1,params=pIIIpars)
[1] 0.002769396 0.013753968 0.061968804 0.238103306 0.676676416 1.000000000
> # calculate quantile function
> qpearsonIII(seq(0.1,0.9,by=0.2),params=pIIIpars)
[1] -1.6611602 -0.8077838 -0.3370302 0.0431121 0.4489673
> # generate random numbers
> rpearsonIII(5,params=pIIIpars)
[1] 0.1961997 -1.5223444 -1.4583283 -0.5992938 -2.3663249
```

۳-۱-۳ نمونه‌گیری تصادفی

تجربه‌های ساده احتمالی مانند انتخاب تصادفی اعداد از 1 تا 100 وکشیدن سه توپ از یک کیسه را می‌توان توسط زبان R شبیه‌سازی نمود. تابع sample() به صورت کلی زیر است.

```
sample(x, size, replace = FALSE, prob = NULL)
```

که در آن،

x: بردار مورد نظر است که می‌تواند عدد و یا کاراکتر باشد.

size: تعدادی که باید انتخاب گردد.

replace: نمونه‌گیری بدون جایگذاری و یا با جایگذاری انجام گردد.

prob: بردار اختیاری است که می‌تواند به انتخاب نمونه‌ها وزن بدهد.

مثال‌ها:

انتخاب یک عدد به تصادف از 1 تا 100

```
> sample(1:100, 1)
[1] 19
```

پرتاب 10 بار یک تاس

```
> sample(1:6, 10, replace = T)
[1] 5 4 3 5 4 6 1 5 2 4
```

پرتاب 10 بار یک تاس با احتمالات نامساوی

```
> sample(1:6, 10, c(0.6,0.4,0.1,0.05,0.03,0.02), replace = T)
[1] 4 2 1 1 1 2 1 1 4 3
```

درکیسه‌ای 8 مهره قرمز، 4 مهره آبی و 3 مهره زرد وجود دارد. اکنون 6 مهره به تصادف و بدون جایگذاری انتخاب کنید.

```
> urn <- c(rep("red", 8), rep("blue", 4), rep("yellow", 3))
> sample(urn, 6, replace = F)
[1] "red" "yellow" "blue" "red" "red" "red"
```

۳-۱-۴ برآورد پارامترهای یک تابع احتمال

همان‌طور که می‌دانید به روش‌های مختلف می‌توان پارامترهای یک تابع احتمال را برآورد نمود. از روش‌هایی مانند روش گشتاورها و روش حداکثر درستنمایی[۶] می‌توان نام برد و به آنها اشاره نمود.

در زبان R بسته‌ای تحت عنوان MASS وجود دارد که پارامترهای تابع احتمال را به روش حداکثر درستنمایی برآورد می‌کند. در این بسته تابعی به نام fitdistr() موجود است که دارای دو آرگومان ضروری است، اول داده‌هایی که قرار است بر آنها تابعی برازش یابد و دوم تابع احتمال مورد نظر که باید در داخل کوتیشن قرار گیرد.

مثال: در اینجا ابتدا یک سری اعداد تصادفی با توزیع گاما (با پارامترهای مشخص) تولید می‌شود. اکنون با فرض تابع احتمال گاما مجدداً پارامترهای تابع احتمال گاما که بر داده‌های مذکور برازش یافته‌است، برآورد می‌گردد.

```
library("MASS")
set.seed(1)
random <- rgamma(500,8.5,2.5)
para <- fitdistr(random,"gamma"); print(para)
para <- unlist(para)
x <- seq(1,10,0.1)
y <- dgamma(x,8.5,2.5)
plot(x,y,type="l")
lines(x,dgamma(x,para[1],para[2]),lty=2)
legend(7.0,0.30,legend=c("exact", "ML"), lty=c(1,2), bg="gray90")
```

با اجرای برنامه بالا شکل ۳-۱۷ حاصل می‌شود.

۳-۱-۵ روش‌های آماری

زبان R میزبان روش‌های آماری و آزمون فرض است. در اینجا روی مشهورترین آن‌ها تاکید می‌گردد.

۳-۱-۵-۱ آزمون یک و دو طرفه t

تابع اصلی برای این نوع آزمون t.test() است. فرض‌ها در اینجا با تابع چگالی احتمال t آزمون می‌شود.

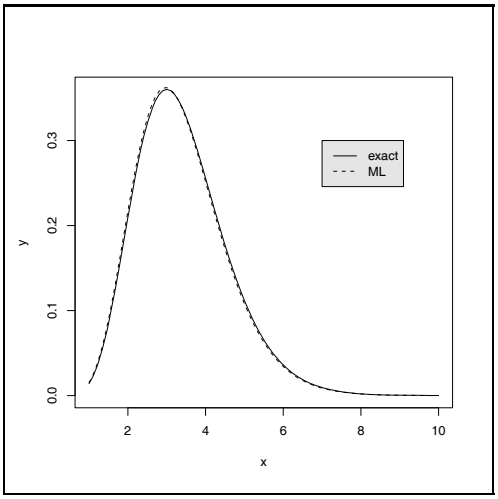
```
t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"), mu = 0,
paired = FALSE, var.equal = FALSE, conf.level = 0.95)
```

که در آن:

x,y: بردارهای عددی هستند. اگر y داده نشود، آنگاه یک آزمون ساده موجود است.

alternative: فرض جایگزین توسط رشته‌ای از کاراکترها بیان می‌گردد. پیش‌فرض آن "two.sided"،

6. Maximum Likelihood



شکل ۳-۱۷: نمایش برازش تابع گاما بر یک سری داده با پارامترهای برآورد شده

"greater" و "less". شما می‌توانید از حروف اول آن نیز برای به‌کارگیری استفاده کنید.
mu: عددی که مقدار درست میانگین را نشان می‌دهد (یا اختلاف میانگین‌ها را بیان می‌کند، اگر دو نمونه وجود داشته باشد). پیش‌فرض برابر صفر است.
paired: نشانه منطقی است اگر بخواهید آزمون دوتایی داشته باشید.
var.equal: متغیر منطقی است. اگر درست (T) باشد واریانس‌ها با هم برابرند و پیش‌فرض آن (F) است.
conf.level: سطح اطمینان (پیش‌فرض 95٪) برای برآورد فاصله‌ای میانگین بر حسب فرض جایگزین

مثال ۱: در داده‌های trees فرض صفر mu=70 را آزمون کنید.

```
> data(trees)
> t.test(trees$Height, mu = 70)

        One Sample t-test

data: trees$Height
t = 5.2429, df = 30, p-value = 1.173e-05
alternative hypothesis: true mean is not equal to 70
95 percent confidence interval:
73.6628 78.3372
sample estimates:
mean of x
       76
```

بنابراین فرض صفر رد می‌شود.

مثال ۲: فرض کنید یک افزودنی به سوخت خودرو اضافه می‌گردد. اکنون سوال این است که آیا از مصرف آن‌ها کاسته می‌شود؟ برای این کار شش خودرو از آن استفاده می‌کند و شش خودروی دیگر از آن استفاده نمی‌کند.

متغیر mpg (مایل بر گالن) اندازه‌گیری شده است، که به‌صورت زیر است.

Car	1	2	3	4	5	6
mpg w/ additive	24.6	18.9	27.3	25.2	22.0	30.9
mpg w/o additive	23.8	17.7	26.6	25.1	21.6	29.6

```
> add <- c(24.6, 18.9, 27.3, 25.2, 22.0, 30.9)
> noadd <- c(23.8, 17.7, 26.6, 25.1, 21.6, 29.6)
> t.test(add, noadd, paired=T, alt = "greater")

        Paired t-test

data: add and noadd
t = 3.9994, df = 5, p-value = 0.005165
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
0.3721225       Inf
sample estimates:
mean of the differences
                  0.75
```

می‌توان یک داده‌ی برداری را با تابع ks.test آزمود.

```
> x <- runif(100)
> out <- ks.test(x, "pnorm")
> out

        One-sample Kolmogorov-Smirnov test

data: x
D = 0.5007, p-value < 2.2e-16
alternative hypothesis: two-sided
```

اگر دو داده‌ی برداری از یک توزیع اخذ شده باشند نیز می‌توان از آزمون k-s استفاده نمود.

```
> x1 <- rnorm(100)
> x2 <- rnorm(100)
> ks.test(x1, x2)

        Two-sample Kolmogorov-Smirnov test

data: x1 and x2
D = 0.08, p-value = 0.9062
alternative hypothesis: two-sided
```

اکنون به یک مثال از آزمون مربع خی توجه فرمایید. فرض کنید که یک تاس 300 بار پرتاب شود و نتایج زیر حاصل گردد.

وجه تاس	1	2	3	4	5	6
فراوانی	43	49	56	45	66	41

اکنون سوال این است که این تاس بی طرف است. یعنی احتمال آمدن هر وجه تاس $\frac{1}{6}$ است؟ برای این کار از آزمون مربع خی استفاده می‌شود. به کُدهای زیر توجه کنید.

```
> counts <- c(43, 49, 56, 45, 66, 41)
> probs <- rep(1/6, 6)
> chisq.test(counts, p = probs)

        Chi-squared test for given probabilities

data: counts
X-squared = 8.96, df = 5, p-value = 0.1107
```

اگر سطح معنی‌دار بودن آزمون فوق 5 درصد باشد ، آنگاه مقدار احتمال یعنی 0.1107 بزرگتر از آن است. بنابراین فرض بی‌طرف بودن رد نمی‌شود.

فصل چهارم

مدل‌های رگرسیون

زبان R دارای روتین‌های زیادی برای برازش مدل‌های آماری است. عموماً این مدل‌ها از طریق فراخواندن توابعی مثل lm, glm, $\cdots$ عمل می‌کنند. شکل کلی یک مدل آماری برازش به صورت زیر است.

respose ~ expression

۱-۴ مدل‌های رگرسیون خطی

۱-۱-۴ موجودیت‌های فرمول

زبان R یک مدل خطی را به صورت زیر برازش می‌دهد.

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \varepsilon$$

که $\beta = (\beta_0, \beta_1, \cdots, \beta_p)$ عرض از مبداء و ضرایب رگرسیون را نشان می‌دهد. جمله خطا یعنی ε غالباً دارای توزیع نرمال با میانگین صفر و واریانس σ_ε^2 است.
برای رگرسیون با دو متغیر، می‌توان با استفاده از تابع lm() و فرمول زیر استفاده نمود.

```
y ~ x1+x2
```

که عبارت فوق به منزله فرمول $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$ است.
به طور پیش‌فرض، زبان R شامل عرض از مبداء می‌شود. اما رابطه فوق را به صورت زیر بنویسید عرض از مبداء حذف می‌گردد.

```
y ~ -1+x1+x2
```

توجه داشته باشید که عملگرهای :,\,^,-,* دارای معانی خاصی در رگرسیون خطی هستند. به مثال‌های زیر توجه کنید.

```
y ~ x1+x2+x1:x2
```

که عبارت فوق به منزله فرمول $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \varepsilon$ است.
استفاده از عملگر ^ جمله‌های یگانه و دودویی را به ترتیب ایجاد می‌کند.

```
y ~ (x1+x2+x1:x2)^2
```

فرمول بالا معادل رابطه زیر است.

```
y ~ x1+x2+x3+x1:x2+x2:x3+x1:x3
```

عملگر - جمله و یا جملاتی را از معادله رگرسیون حذف می‌کند. همان‌طور که قبلاً ملاحظه شد 1- عرض از مبداء را حذف نمود. به مثال زیر توجه کنید.

```
y ~ (x1+x2+x1:x2)^2 - x2:x3
```

فرمول بالا معادل رابطه زیر است.

```
y ~ x1+x2+x3+x1:x2+x1:x3
```

تابع I ضریبی را برای متغیر تکرار می‌کند. به مثال زیر توجه کنید.

```
y ~ I(x1+x2)
```

که عبارت فوق به منزله فرمول $y = \beta_0 + \beta(x_1 + x_2) + \varepsilon$ است. ضمناً اگر بخواهید متغیر x2 در مدل تغییر کند. مثلاً در دو ضرب شود. اگر به صورت زیر عمل کنید اشتباه کرده‌اید.

```
y ~ x1+2*x2
```

برای درست شدن مطلب به صورت زیر باید عمل نمود.

```
y ~ x1+I(2*x2)
```

اگر مدل مورد نظر به‌صورت $y = \beta_0 + \beta_1 x + \beta_2 x^2$ باشد، آنگاه فرمول آن در زبان R به شرح زیر است.

```
y ~ poly(x, 2)
```

و نه به‌صورت زیر:

```
y ~ x+x2^2
```

۴-۱-۲ توابع مدل‌سازی

مدل‌های رگرسیون خطی کاربرد وسیعی برای بیان روابط خطی بین متغیرها دارد. توابع زیادی برای برازش و تحلیل رگرسیون خطی وجود دارد. تابع اصلی برای این کار lm() است که پاره‌ای از آرگومان‌های آن‌ها به شرح زیر است.

lm(formula, data, weights, subset, na.action)

به داده‌های زیر توجه کنید.

gene1	gene2
-1.06	-1.08
-0.81	-1.02
-0.48	-0.39
-0.42	-0.48
-0.30	-0.58
-0.35	-0.24
-0.31	-0.05
-0.18	-0.33
-0.20	0.51
-0.11	-0.53
-0.09	-0.47
0.16	0.10
0.45	0.39
0.53	0.11
0.67	0.52
0.80	0.34
0.87	1.08
0.92	1.21

داده‌های فوق در فایلی به نام gen_reg.txt ذخیره شده است. که ابتدا ضریب همبستگی و آزمون مربوط به آن را ملاحظه خواهید نمود. کُدهای زیر آن را خوانده موارد مربوط به آن‌ها را بررسی می‌کند.

```
> x <- read.table("F:/R_files/data/gen_reg.txt",header=T)
> cor.test(x$gene1, x$gene2)

        Pearson's product-moment correlation

data: x$gene1 and x$gene2
```

```
t = 7.5105, df = 16, p-value = 1.246e-06
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
0.7071560 0.9556856
sample estimates:
      cor
0.8826268
```

اکنون مدل رگرسیون با فرمول زیر

$$\text{gene2} = \beta_0 + \beta_1 \times \text{gene1} + \varepsilon$$

روی داده‌ها اعمال می‌گردد.

```
> x <- read.table("F:/R_files/data/gen_reg.txt", header=T)
> m.reg <- lm(gene2~gene1, data=x)
> m.reg

Call:
lm(formula = gene2 ~ gene1, data = x)

Coefficients:
 (Intercept)     gene1
   -0.05541   0.97070
```

کُدهای بالا را می‌توان به‌صورت زیر هم نیز نوشت.

```
> x <- read.table("F:/R_files/data/gen_reg.txt", header=T)
> m.reg <- lm(x$gene2~x$gene1)
> m.reg

Call:
lm(formula = x$gene2 ~ x$gene1)
Coefficients:
 (Intercept)     gene1
   -0.05541   0.97070
```

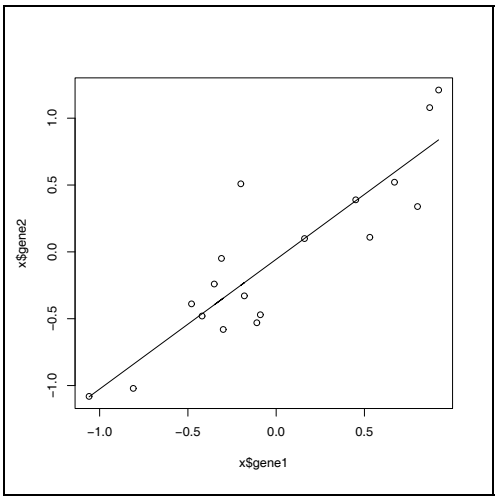
برای یافتن خروجی‌های مربوط به رگرسیون به جدول زیر توجه کنید.

Expression	Description
coef(obj)	regression coefficients
resid(obj)	residuals
fitted(obj)	fitted values
summary(obj)	analysis summary
predict(obj,newdata=ndat)	predict for new data
deviance(obj)	residual sum of squares

در مثال فوق منظور از obj همان m.reg است.
اکنون می‌توان نمودار پراکنش را رسم نمود و خط برازش یافته را کشید.

```
> plot(x$gene1, x$gene2)
> lines(x$gene1, fitted(m.reg))
```

با اجرای کُدهای زیر شکل ۴-۱ حاصل می‌شود. برای رسم خط برازش یافته می‌توان از دستورات زیر نیز استفاده



شکل ۴-۱: نمایش یک نمودار پراکَنش و خط برازش یافته

نمود.

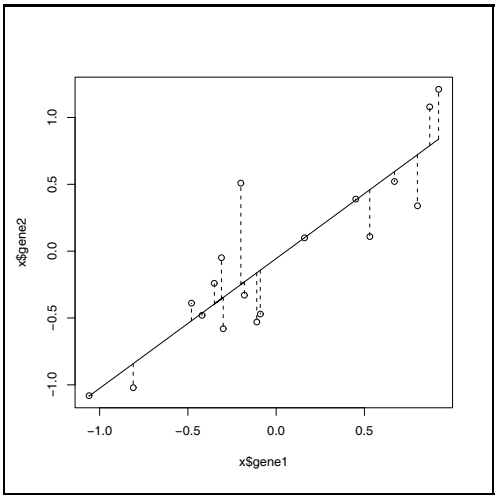
```
> plot(x$gene1, x$gene2)
> abline(m.reg)
```

اکنون در زیر شکل ۴-۲ را ملاحظه می‌کنید که علاوه بر نقاط تجربی و خط برازش یافته، باقیمانده‌ها را نیز می‌توان رسم نمود.

```
> plot(x$gene1, x$gene2)
> lines(x$gene1, fitted(m.reg))
> segments(x$gene1, fitted(m.reg), x$gene1, x$gene2, lty=2)
```

تابع summary() برای بدست آوردن پاره‌ای از اطلاعات اضافی از مدل برازش یافته نظیر مقادیر t، خطای استاندارد و همبستگی بین پارامترها مفید است. به مثال زیر توجه کنید.

```
> summary(m.reg)

Call:
lm(formula = x$gene2 ~ x$gene1)

Residuals:
     Min       1Q   Median       3Q      Max
-0.3812  -0.2196  -0.0084   0.1492   0.7595
```



شکل ۴-۲: نمایش یک نمودار رگرسیون و باقیمانده‌ها

Coefficients:

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	-0.05541	0.07330	-0.756	0.461	
x$gene1	0.97070	0.12925	7.511	1.25e-06	***

—

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.311 on 16 degrees of freedom
Multiple R-squared: 0.779, Adjusted R-squared: 0.7652
F-statistic: 56.41 on 1 and 16 DF, p-value: 1.246e-06

با تابع confint() می‌توان برآورد فاصله‌ای ضرایب معادله رگرسیون را محاسبه نمود.

```
> coef(m.reg)
  (Intercept)         gene1
-0.05540906    0.97070027
> confint(m.reg)
                    2.5%        97.5%
(Intercept)   -0.2107882   0.09997012
gene1          0.6967126   1.24468796
```

تابع summary() به‌صورت بردار، اطلاعاتی را در خود ذخیره می‌کند که می‌توان آنها را به‌صورت انفرادی نیز نشان داد و از آنها در قسمت‌های دیگر استفاده نمود. اکنون به موارد زیر توجه کنید.

```
summary(model)[3]
$residuals
summary(model)[4]
```

```
$coefficients
summary(model)[6]
$sigma
summary(model)[8]
$r.squared
summary(model)[9]
$adj.r.squared
summary(model)[10]
$fstatistic
```

همانطورکه مشاهده می‌کنید در اینجا نام مدل m.reg است. یعنی به جای واژه مدل m.reg قرار می‌گیرد.
برای به‌دست آوردن خط رگرسیون و مشخصاتی که تابع summary() به‌دست می‌دهد، می‌توان از تابع with نیز استفاده نمود. به مثال زیر توجه کنید.

```
> with(trees, {m.reg <- lm(Girth~Volume)
+ summary(m.reg)})
```

با اجرای کُدهای بالا همان نتایج روش قبل حاصل می‌شود. حتی در مورد ترسیم نقاط و خط برازش یافته هم می‌توان از تابع with استفاده نمود. به کُدهای زیر توجه کنید.

```
> with(trees, {plot(Girth~Volume)
+ abline(lm(Girth~Volume))})
```

در اینجا لازم است که راجع به R^2_{adj} یعنی ضریب تعیین تعدیل شده توضیحی آورده شود تا معلوم گردد که تفاوت آن با R^2 چیست؟ همانطورکه می‌دانید میزان بالای مقدار ضریب تعیین در همبستگی درخور توجه است. اما این مقدار با تعداد متغیرها و همچنین تعداد داده‌ها تغییر می‌کند. بنابراین نمی‌توان مدل‌ها را با هم مقایسه نمود. لذا مقدار ضریب تعیین تعدیل شده پیشنهاد شده است. تعریف R^2 به‌صورت زیر است.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = 1 - \frac{SS_e}{SS_t}$$

که در آن $SS_e = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ و $SS_t = \sum_{i=1}^{n}(y_i - \bar{y})^2$ می‌باشند. با دخالت درجه آزادی‌ها می‌توان ضریب تعیین تعدیل یافته را تعریف نمود.

$$R^2_{adj} = 1 - \frac{SS_e/(n-p)}{SS_t/(n-1)} = 1 - \left(\frac{n-1}{n-p}\right)\frac{SS_e}{SS_t}$$

که در آن n تعداد نمونه‌ها و p تعداد متغیرها است. البته تعداد متغیر مستقل برابر $p-1$ می‌باشد.
می‌توان مقادیر جدیدی را با استفاده از معادله رگرسیون برآورد نمود. برای این کار از تابع predict.lm() می‌توان بهره گرفت. در این صورت هم مقدار برآورد نقطه‌ای و هم برآورد فاصله‌ای را بدست داد.

```
> predict.lm(m.reg, newdata=data.frame(gene1=c(0.1,0.2)), int="conf")
```

	fit	lwr	upr
1	0.04166097	-0.11587731	0.1991993
2	0.13873100	-0.02557153	0.3030335

اگر از newdata استفاده نشود، آنگاه تمام yهای نقطه‌ای و برآورد فاصله‌ای آنها به ازای تمام xهای مشاهده شده بدست می‌آید.
برای مقادیر جدید می‌توان به صورت زیر نیز عمل نمود.

```
> predict(m.reg, list(gene1=c(0.1,0.2)), int="conf")
```

	fit	lwr	upr
1	0.04166097	-0.11587731	0.1991993
2	0.13873100	-0.02557153	0.3030335

۴-۱-۲-۱ تشخیص مدل

برای صحت سنجی مدل بدست آمده، می‌توان از آزمون‌ها و شاخص‌هایی استفاده نمود. برای مثال می‌توان به موارد زیر اشاره نمود.

- آیا باقیمانده‌ها نرمال هستند؟
- آیا رابطه بین پاسخ و متغیرهای رگرسیون خطی هستند؟
- آیا مقادیر پرت وجود دارد؟

آزمون k-s توزیع نرمال باقیمانده‌ها را کنترل می‌کند.

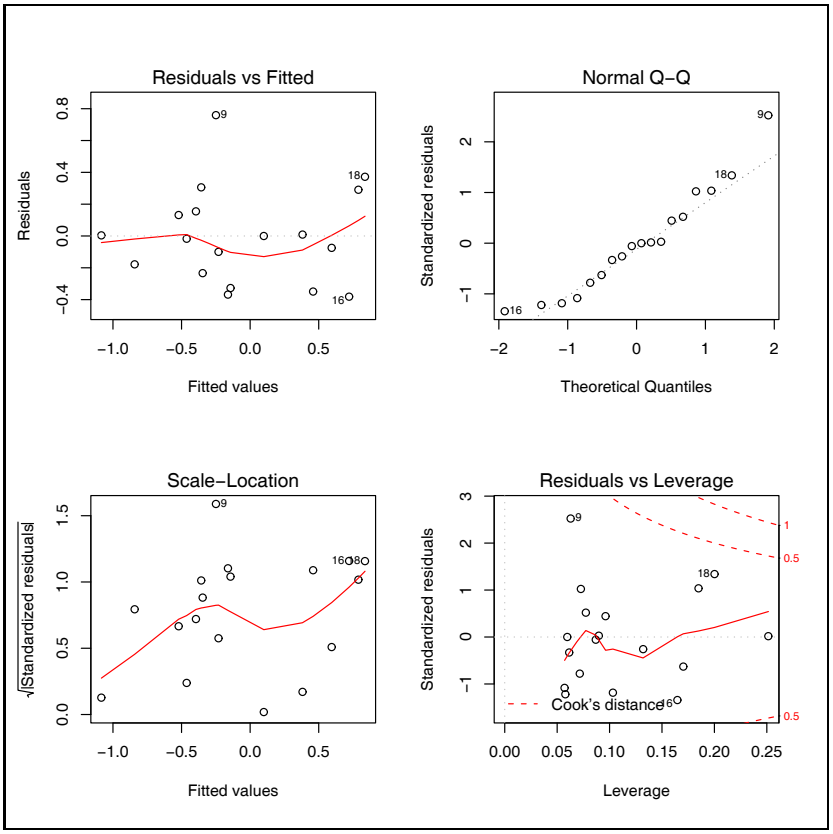
```
> res <- resid(m.reg)
> ks.test(res, "pnorm", mean=mean(res), sd=sd(res))

        One-sample Kolmogorov-Smirnov test

data: res
D = 0.1553, p-value = 0.7217
alternative hypothesis: two-sided
```

اگر دستورات زیر را در دنباله کُدهای قبلی بیاورید، چهار نمودار شکل ۴-۳ ظاهر می‌گردد.

```
> par(mfrow=c(2,2))
> plot(m.reg)
```

این چهار نمودار عبارتست از:

- مقادیر باقیمانده‌ها بر حسب مقادیر برازش یافته: شکل حاصل باید حول خط y=0 قرارگیرد و از خود روندی را نشان ندهد.
- نمودار qqplot نرمال: اگر نقاط باقیمانده نزدیک به نمایش یک خط مستقیم شوند، بدین معنی است که نمودار باقیمانده از توزیع نرمال پیروی می‌کند.
- نمودار scale-location: این نمودار ریشه دوم باقیمانده‌های استاندارد شده را نشان می‌دهد. نقاط بلند، بزرگترین مقادیر باقیمانده هستند.



شکل ۴-۳: نمایش نمودارهای مربوط به باقیمانده‌ها

- نمودار Cook's distance: این نمودار نقاطی راکه روی خط رگرسیون تأثیر زیادی دارند را نشان می‌دهد، که می‌تواند به‌عنوان نقاط پرت نیز تلقی گردد.

برای رسم نمودارهای بالا به صورت انفرادی و حتی نمودارهای بیشتر، می‌توان از دستور زیر استفاده نمود.

```
plot(m.reg, which = 1)
```

که اولین شکل را رسم می‌کند. برای ترسیم نمودار qqplot درکد اخیر بجای عدد 1 از عدد 2 استفاده می‌شود.

```
plot(m.reg, which = 2)
```

برای رسم نمودارها می‌توان تا عدد 6 نیز پیش رفت.

۴-۱-۲-۲ فاصله اطمینان

برای ایجاد فاصله اطمینان می‌توان از کُدهای زیر استفاده نمود.

```
x <- read.table("E:/R_files/data/gen_reg.txt", header=T)
m.reg <- lm(x$gene2~x$gene1)
new <- data.frame(x$gene1 <- sort(x$gene1))
pred.w.clim = predict(m.reg, newdata=new, interval="confidence",level=0.95)
plot(x$gene2~x$gene1, pch=20)
abline(m.reg, col="blue")
lines(x$gene1,pred.w.clim[,2],col="red", lty=2)
lines(x$gene1,pred.w.clim[,3],col="red", lty=2)
pred = predict(m.reg, interval="confidence"); print(pred)
pred1 = predict(m.reg, interval="confidence")[,1,drop=F]; print(pred1)
legend(-1, 1.2,
c("obs. data", "predict", "Lower b.", "Upper b."),
pch=c(20,NA,NA,NA), lty=c(0,1,2,2), lwd=c(0,1,1,1),
col=c("black","blue","red","red"))
```

با اجرای کُدهای فوق شکل ۴-۴ بدست می‌آید.

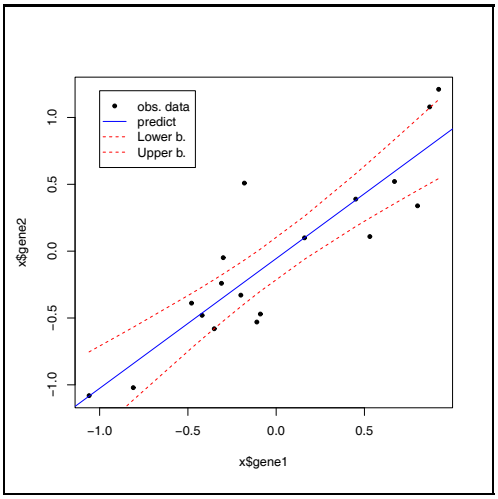
برای مقایسه بین مدل‌های برازش یافته می‌توان از ضابطه‌ای استفاده نمود که AIC[1] نامیده می‌شود. مقدار کمتر آن مناسب‌تر است. برای مثال از داده‌ای trees استفاده می‌شود که در خود R موجود است. برای مشاهده آن که شامل سه ستون است کافی است که دستور زیر را اجرا کنید.

```
> trees
```

اکنون دو مدل برای داده‌های trees در نظر گرفته می‌شود و برای هر یک ضابطه AIC محاسبه می‌گردد که می‌توانید با هم آن‌ها را مقایسه کنید.

```
> lm1 <- lm(Volume ~ Height * Girth, data=trees)
> AIC(lm1)
[1] 155.4692
> lm2 <- lm(Volume ~ Height + Girth, data=trees)
```

1. **A**kaike's An **I**nformation **C**riterion



شکل ۴-۴: نمایش یک نمودار رگرسیون با فاصله اصمینان ۹۵٪

```
> AIC(lm2)
[1] 176.9100
```

۴-۱-۲-۳ رسم صفحه نمودار

اکنون که بحث داده‌های مجموعه trees شد و مدل‌هایی از آن با هم مقایسه گردید، مناسب است که نمودار ۴-۵ که صفحه رگرسیون است نیز ترسیم گردد. کُدهای مورد نظر به‌صورت زیر است.

```
> library(scatterplot3d)
> s3d <- with(trees, scatterplot3d(Girth, Height, Volume, pch = 16,
+ highlight.3d = TRUE, angle = 60))
> fit <- lm(Volume ~ Girth + Height, data = trees)
> s3d$plane3d(fit)
```

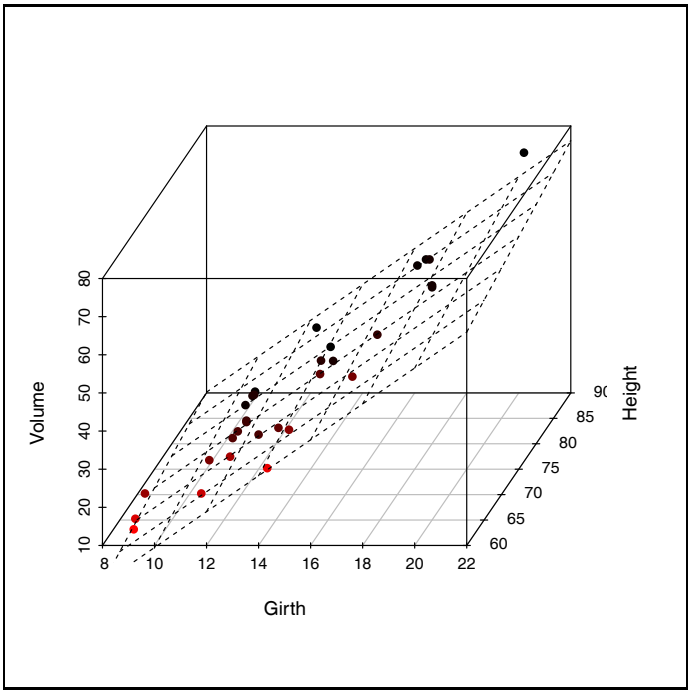
۴-۱-۳ داده‌های گمشده در رگرسیون

در رگرسیون ممکن است که زوج مرتب[2]هایی ناقص باشند، یعنی یکی از مؤلفه‌های آنها گمشده باشند. در زبان R بخوبی این امر برای به‌دست آوردن رگرسیون قابل مدیریت است. به مثال زیر توجه کنید.

```
> x <- c(12,4,6,8,13,15,19)
> y <- c(11,2,5,NA,20, 23, 33)
> z <- c(3,6,8,11,40,21,99)
> reg1 <- lm(y~x+z)
> reg1
```

پس از اجرای کُدهای اخیر نتیجه به‌صورت زیر است.

2. pairwise



شکل ۴-۵: نمودار صفحه رگرسیون

```
Call:
lm(formula = y ~ x + z)

Coefficients:
(Intercept)            x            z
 -5.19862      1.57928      0.09165
```

و اگر در ادامه دستور زیر اجرا شود، نتیجه به‌صورت زیر است.

```
> reg1$model

 y  x  z
1 11 12  3
2  2  4  6
3  5  6  8
5 20 13 40
6 23 15 21
```

```
7 33 19 99
```

همانطورکه ملاحظه می‌شود زوج شماره 4 حذف شده است.
اکنون رگرسیون با همان داده‌ها منتها با دو زوج ناقص اجرا می‌شود.

```
> x <- c(12,4,6,8,13,15,19)
> y <- c(11,2,5,NA,20, 23, 33)
> z <- c(3,6,8,11,40,21,NA)
> reg2 <- lm(y~x+z)
> reg2


Call:
lm(formula = y ~ x + z)


Coefficients:
(Intercept)            x            z
 -5.6706       1.4996       0.1843

> reg2$model


 y  x  z
1 11 12  3
2  2  4  6
3  5  6  8
5 20 13 40
6 23 15 21
```

همانطورکه ملاحظه می‌شود زوج‌های شماره 4 و 7 حذف شده است.

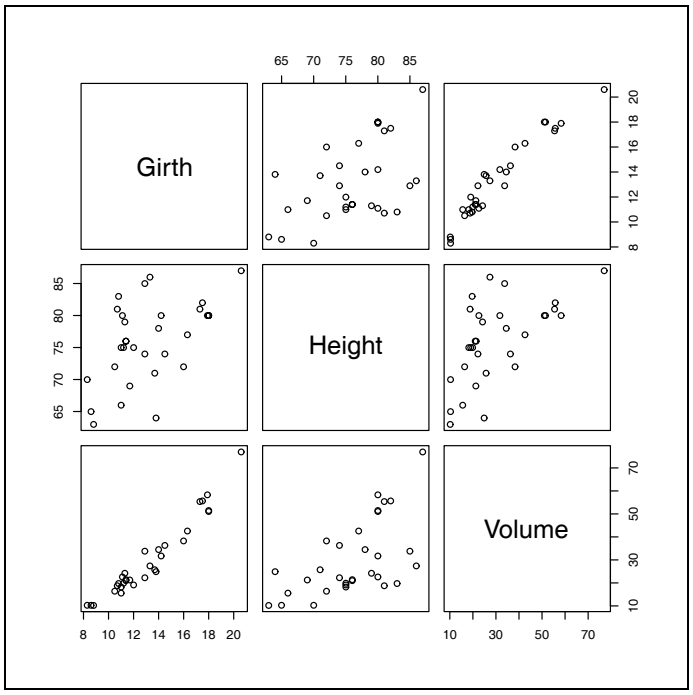
۴-۱-۳-۱ رسم نمودارهای گروه‌های یک داده

همان‌طورکه تاکنون ملاحظه شد می‌توان داخل یک داده گروه‌های مختلفی داشت. در پاره‌ای از اوقات ترسیم آنها نسبت یکدیگر می‌تواند برای کاربر مفید باشد. این کار در R با استفاده از تابع pairs() مقدور است و در واقع این تابع یک ماتریسی از نمودارهای پَراکنش را به‌دست می‌دهد. برای مثال از داده‌های trees در R استفاده می‌شود.

```
pairs(trees)
```

نتیجه شکل ۴-۶ خواهد بود.

در ماتریس نمودار پَراکنش می‌توان به کمک رگرسیون، خطوط برازش یافته بر هر نمودار را به‌صورت مجزا نیز



شکل ۴-۶: نمایش نمودار گروه‌های یک داده نسبت به‌هم

داشت. برای این کار در تابع ()pairs آرگومانی به نام panel وجود دارد که می‌توان در آن تابع مورد نظر خود را تعریف نمود. اکنون به مثال زیر توجه کنید.

```
> pairs(trees, panel=function(x,y) {points(y~x); abline(lm(y~x))})
```

حاصل اجرای دستور فوق شکل ۴-۷ خواهد بود.

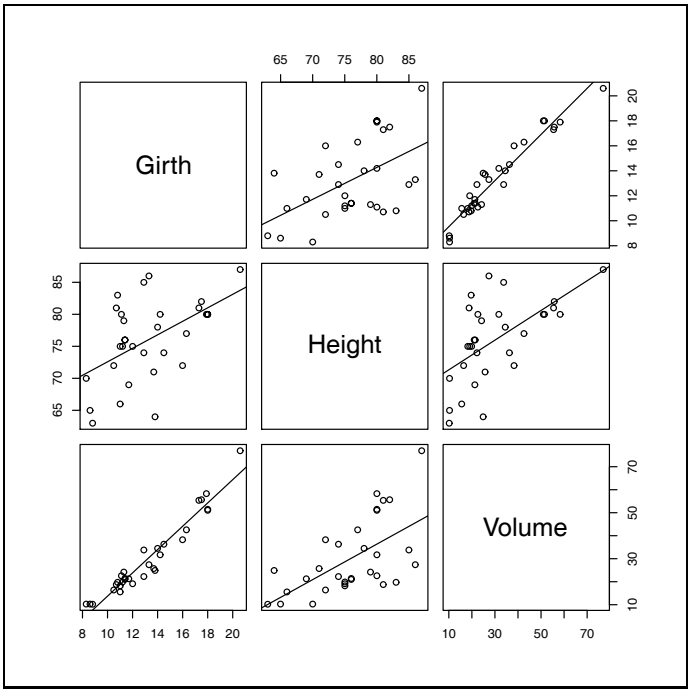
البته برازش فوق را می‌توان به‌طور ناپارامتری نیز داشت. به مثال ناپارامتری داده trees توجه کنید.

```
> pairs(trees, panel=panel.smooth)
```

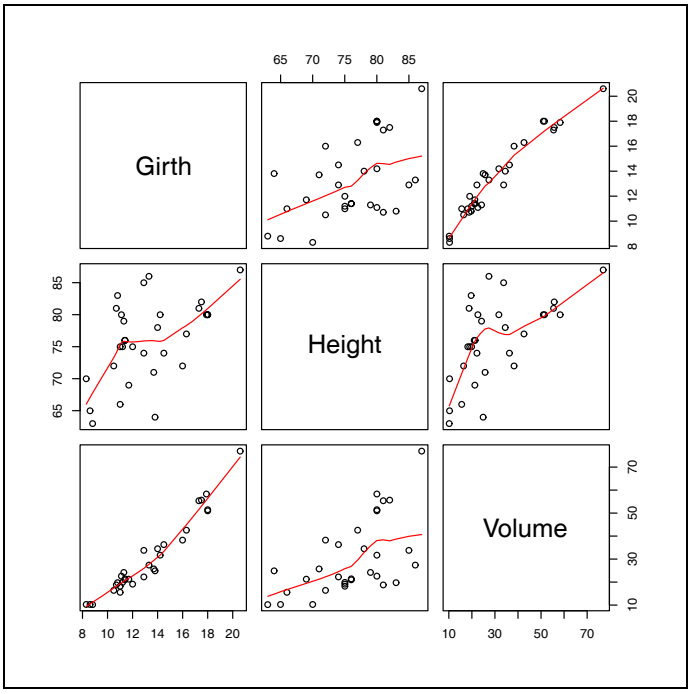
حاصل اجرای دستور فوق شکل ۴-۸ خواهد بود.

۴-۲ درون‌یابی، هموارسازی و برازش منحنی

در این نوع تکنیک‌ها معمولاً بحث برازش‌های غیرخطی عنوان می‌شود و شیوه‌های مختلفی نیز برای آنها بیان شده است. شرح پاره‌ای از آنها ذیلاً تقدیم می‌گردد.



شکل ۴-۷: نمایش نمودار گروه‌های یک داده و خط برازش یافته



شکل ۴-۸: نمایش نمودار گروه‌های یک داده و برازش ناپارامتری

۴-۲-۱ درون‌یابی و هموارسازی

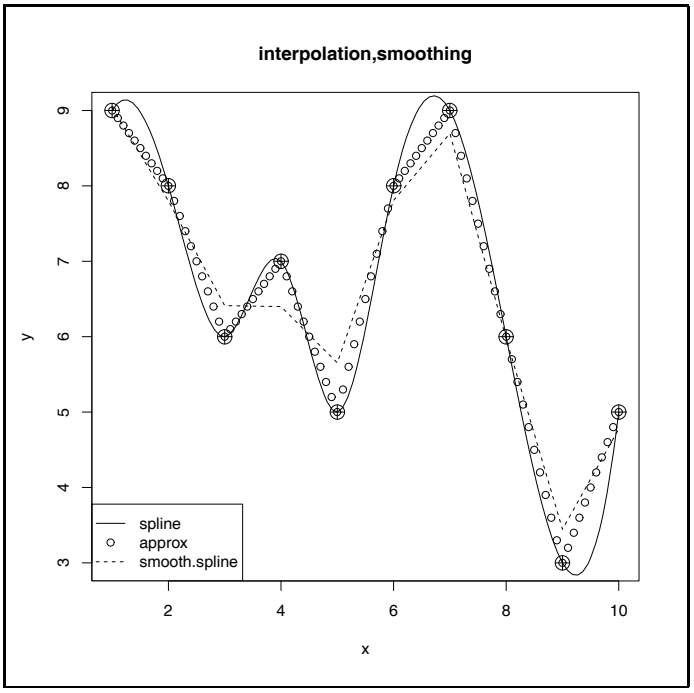
دورن‌یابی و هموارسازی در R به‌صورت‌های مختلفی قابل اعمال هستند که بعضی از آنها را در زیر مشاهده می‌کنید.

- درون‌یابی خطی (تابع approx()) که بین نقاط انجام می‌شود.
- درون‌یابی با استفاده از مکانیزم تابع spline() که هموارتر از روش قبلی است.
- مجموعه نقاطی که قدری هموارتر (تابع smooth.spline()) شده است اما نقاط اصلی را به هم متصل نمی‌کند.

اکنون به مثال زیر توجه کنید.

```
> x <- 1:10
> y <- c(9,8,6,7,5,8,9,6,3,5)
> plot(x,y, pch=10, cex=2, main="interpolation,smoothing")
> lines (spline(x,y, n=100), lty=1)
> points(approx(x,y, xout=seq(1,10,0.1)), pch=1)
> lines (smooth.spline(x,y), lty=2)
> legend("bottomleft",lty=c(1,NA,2), pch=c(NA,1,NA),
+ legend=c("spline","approx","smooth.spline"))
```

با اجرای کُدهای بالا شکل ۴-۹ نتیجه می‌شود.



شکل ۴-۹: نمایش نمودارهای درون‌یابی و هموار شده

۴-۲-۲ برازش منحنی

زبان R دارای چندین رَویه برازش منحنی است. بر حسب این که تابع برازش یافته خطی و یا غیرخطی باشد، می‌توان از موارد زیر استفاده نمود.

- توابع lm و glm به ترتیب برای مدل‌های خطی و مدل‌های غیرخطی تعمیم یافته به‌کار برده می‌شود.
- توابع nls، nlm، optim و costrOptim برای مدل‌های غیرخطی استفاده می‌شود.

مثال: رشد جمعیت انسانی (N میلیون نفر) در زمان مشخص t می‌تواند تابعی از زمان تعریف گردد. جمعیت اولیه در زمان t_0 برابر N_{t_0}، پارامترهای K حداکثر جمعیت مورد انتظار و a برابر نرخ رشد است. مدل کلی (غیرخطی) افزایش جمعیت به‌صورت زیر است.

$$N(t) = \frac{K}{1 + \left(\frac{K - N_{t_0}}{N_{t_0}}\right) e^{-a(t-t_0)}}$$

جمعیت کشوری در سال 1900 برابر $N_{t_0} = 76.1$ میلیون نفر است. پارامترهای a و K به ترتیب برابر 0.02 و 500 است. جمعیت واقعی در سال‌های مختلف به‌صورت جدول زیر است.

1900	1910	1920	1930	1940	1950	1960	1970	1980
76.1	92.4	106.5	123.1	132.6	152.3	180.7	204.9	226.5

برنامه با داده‌های ورودی زیر آغاز می‌گردد.

```
year <- seq(1900, 1980, by=10)
pop <- c(76.1,92.4,106.5,123.1,132.6,152.3,180.7,204.9,226.5)
```

ساده‌ترین روش برازش منحنی غیر خطی در R، تابع nls() است. این تابع به‌صورت $y \sim f(x, \text{parameters})$ فرموله می‌شود و مقادیر اولیه پارامترها باید مشخص گردد.

در این مثال y مقادیر جمعیت، f مدل رشد است و شرایط اولیه $K = 500$، $N_{t_0} = 76.1$ و $a = 0.02$ است.

```
> year <- seq(1900, 1980, by=10)
> pop <- c(76.1,92.4,106.5,123.1,132.6,152.3,180.7,204.9,226.5)
> fit <- nls(pop~K/(1+(K-N0)/N0*exp(-a*(year-1900))),
+ start=list(K=500, N0=76.1, a=0.02))
> summary(fit)
```

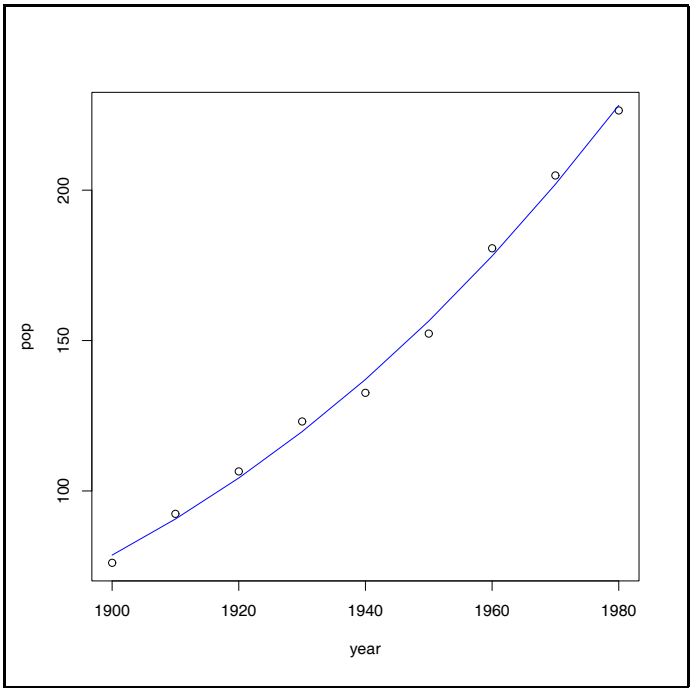
با اجرای کُدهای بالا نتایج زیر حاصل می‌شود.

```
Formula: pop ~ K/(1 + (K - N0)/N0 * exp(-a * (year - 1900)))

Parameters:
 Estimate Std. Error t value Pr(>|t|)
K  1.008e+03  8.932e+02   1.129  0.30210
```

```
N0 7.866e+01  2.531e+00  31.084 7.36e-08 ***
a  1.550e-02  2.505e-03   6.188  0.00082 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.685 on 6 degrees of freedom

Number of iterations to convergence: 6
Achieved convergence tolerance: 4.301e-06
```

برای ترسیم نمودار (شکل ۴-۱۰) مربوطه، کُدها را به‌صورت زیر ادامه دهید.

```
> plot(pop~year)
> K <- coef(fit)[1]; N0 <- coef(fit)[2]; a <- coef(fit)[3]
> model <- K/(1+(K-N0)/N0*exp(-a*(year-1900)))
> lines(model~year, col="blue")
```



شکل ۴-۱۰: نمایش نمودار جمعیت و مدل برازش یافته

فصل پنجم

ترسیم چند نمودار فنی و تخصصی

در این قسمت شما چند نمودار پیچیده به اضافه کُدهای مربوطه را ملاحظه می‌کنید. باشد تا از این رهگذر ایده‌های مناسبی جهت ترسیم نمودارهای فنی برای خوانندگان گرامی فراهم گردد. قبل از ارایه آنها ذکر این نکته ضروری است که کُدهای مربوطه به صورت اسکریپ ارایه می‌شود.

۵-۱ ترسیم کاغذهای احتمال

کاغذهای احتمال برای برازش داده‌های هواشناسی و هیدرولوژی کاربرد فراوانی دارد. در این کاغذها یکی از محورها دارای مقیاس احتمالی است و محور دیگر خطی و یا لگاریتمی است. این کاغذها انواع گوناگون دارد که می‌توان از کاغذهای نرمال و لوگ نرمال و گامبل نام برد. توجه داشته باشید که این کاغذها را نباید با کاغذ نیمه گاریتمی خَلْط نمود. در این ترسیم‌ها از کلیه اطلاعات، دستورات و توابع فصول گذشته استفاده گردیده است.

۵-۱-۱ کاغذ احتمال گامبل

در اینجا کاغذ احتمال گامبل و توزیع برازش یافته بر داده‌های مشاهده شده را ملاحظه می‌کنید. در بادی امر کدهای مورد نظر به‌صورت زیر است.

```
# GumbelPlot.R
#
# Code for plotting annual peak flow series on
# extreme-value (Gumbel) paper.
# # This code illustrates how to customize graph axes, and
# also how to use superscripts in axis labels.
#
# RDM 2007 Oct 25

# Specify Greata Creek peak flows for 1971 - 1980
Q = c(1.23,2.37,0.085,1.69,1.2,0.898,0.176,0.96,0.212,0.266)
graphlab = "Greata Creek 1971-1980"
# Generate plotting positions
n = length(Q)
r = n + 1 - rank(Q) # highest Q has rank r = 1
T = (n + 1)/r
# Set up x axis tick positions and labels
Ttick = c(1.001,1.01,1.1,1.5,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,25
,30,35,40,45,50,60,70,80,90,100)
xtlab = c(1.001,1.01,1.1,1.5,2,NA,NA,5,NA,NA,NA,NA,10,NA,NA,NA,NA,15,
NA,NA,NA,NA,20,NA,30,NA,NA,NA,50,NA,NA,NA,NA,100)
y = -log(-log(1 - 1/T))
ytick = -log(-log(1 - 1/Ttick))
xmin = min(min(y),min(ytick))
xmax = max(ytick)
# Plot peak flow series with Gumbel axis
plot( y, Q, ylab = expression( "Peak Flow ("*m^3*s^-1*")" ) ,
xaxt = "n", xlab = "T (yr)",
xlim = c(xmin,xmax),
pch = 21, bg = "red",
main = paste( "Annual Peak Flows,", graphlab ))
par(cex = 0.65)
axis(1, at = ytick, labels = as.character(xtlab))
# Draw a best-fit line through the data
abline( lm(Q   y), lty=1, col = "blue" )
# Draw grid lines
abline(v = ytick, lty = 3)
abline(h = c(0.5,1,1.5,2), lty = 3)
par(cex = 1)
```

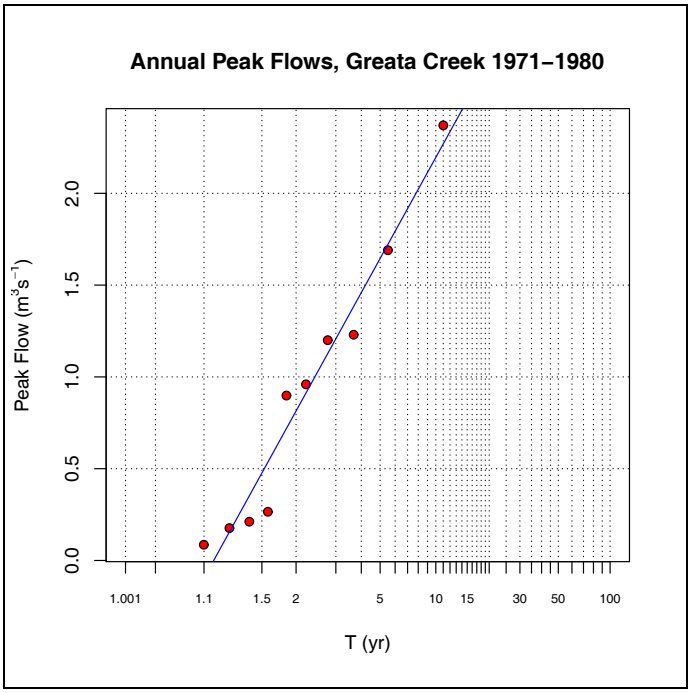
با اجرای کُدهای فوق شکل ۵-۱ حاصل می‌گردد.

۵-۱-۲ ترسیم کاغذ شدت-مدت-فراوانی

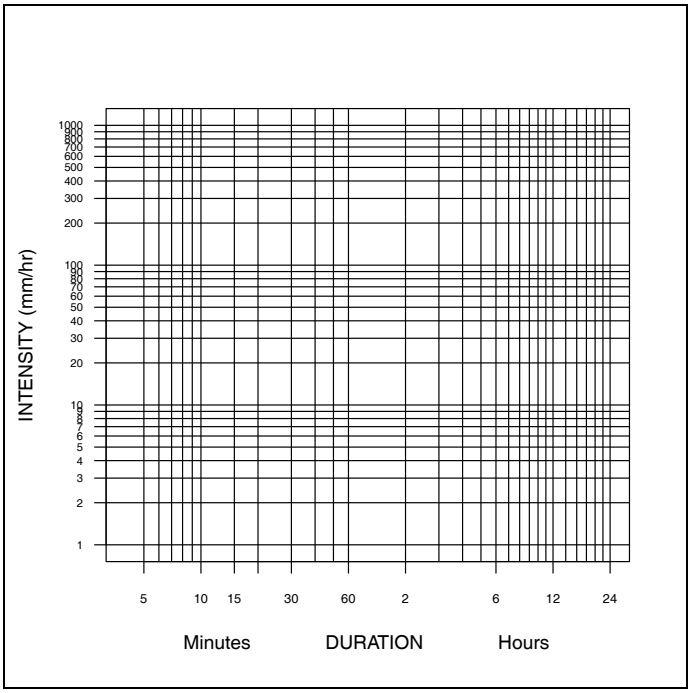
این کاغذ برای رسم منحنی‌های شدت-مدت-فراوانی بارندگی (رگبارها) استفاده می‌شود. البته واضح است که شکل منحنی‌ها به‌صورت خط است. زیرا یک محور شکل لگاریتمی است. کُدهای آن به‌صورت زیر است.

```
# Graph paper for IDF analysis
#
# This code illustrates how to customize graph axes and
# how to produce grid lines.
#
# RD Moore 2007-Jan-23
#
par(cex=1,las=1)
ytick = c(1,2,3,4,5,6,7,8,9,10,20,30,40,50,60,70,80,90,100,
200,300,400,500,600,700,800,900,1000)
yticklab = as.character(ytick)

xgrid = c(5,6,7,8,9,10,15,20,30,40,50,60,120,180,240,300,360,
420,480,540,600,660,720,840,960,1080,1200,1320,1440)
xtick = c(5,10,15,20,30,60,120,360,720,1440)
xticklab = c("5","10","15","20","30","60","2","6","12","24")

x = 5
y = 10

plot( x, y, type="n",
xaxt="n",yaxt="n",
log="xy",
xlim=c(4,1440),
ylim=c(1,1000),
ylab = "INTENSITY (mm/hr)",
xlab = "Minutes DURATION Hours")

par(cex=0.65)
axis(1, xtick, xticklab)

par(cex=0.6)
axis(2, ytick, yticklab)

abline(v=xgrid)
abline(h = ytick)

par(cex=1)
```

با اجرای کُدهای فوق شکل ۵-۲ حاصل می‌گردد.



شکل ۵-۱: نمایش کاغذ گامبل و توزیع برازش یافته



شکل ۵-۲: نمایش کاغذ شدت-مدت-فراوانی

۵-۱-۳ ترسیم رگرسیون همراه با توابع حاشیه‌ای

رسم نمودار پراکنش همراه با خط رگرسیون کار دشواری نیست. اما اگر بخواهید که توابع حاشیه‌ای دو متغیر نیز روی شکل موجود رسم گردد، نموداری جالبی بوجود می‌آید. کُدهای آن به‌صورت زیر است.

```
## Create a scatterplot with marginal histograms ##
x <- pmin(3, pmax(-3, rnorm(75)))
y <- pmin(3, pmax(-3, rnorm(75)))
xhist <- hist(x, breaks=seq(min(x),max(x),length=11) ,plot=FALSE)
yhist <- hist(y, breaks=seq(min(y),max(y),length=11), plot=FALSE)
top <- max(c(xhist$counts, yhist$counts))
xrange <- c(min(x),max(x))
yrange <- c(min(y),max(y))
nf <- layout(matrix(c(2,0,1,3),2,2,byrow=TRUE), c(3,1), c(1,3), TRUE)
layout.show(nf)
par(mar = c(3,3,1,1))
plot(x, y, xlim=xrange, ylim=yrange, xlab="", ylab="",col=6, cex=1.5)
par(mar = c(0,3,1,1))
grid(col=3)
abline(lm(y~x-1), col="brown",lwd=2)
barplot(xhist$counts, axes=FALSE, ylim=c(0, top), space=0, col="red")
par(mar = c(3,0,1,1))
barplot(yhist$counts, axes=FALSE, xlim=c(0, top), space=0, horiz=TRUE,
col="blue")
```

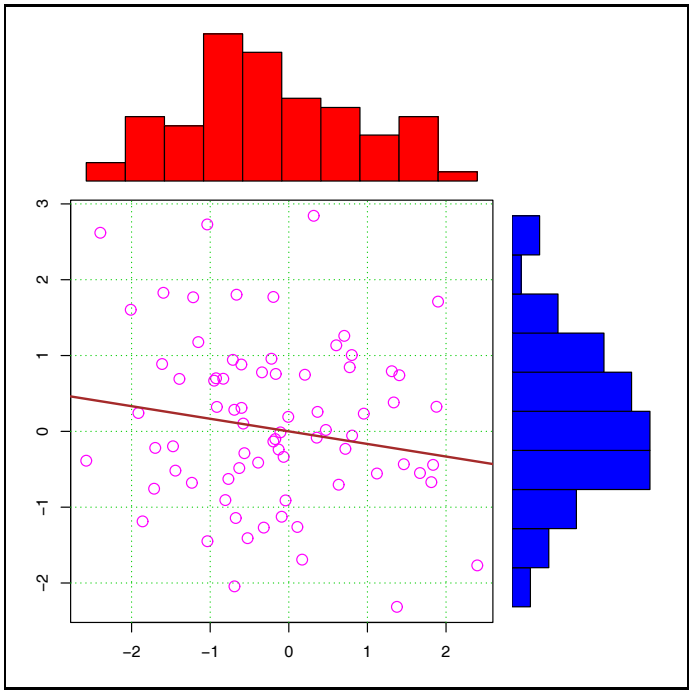
با اجرای کُدهای فوق شکل ۵-۳ حاصل می‌گردد.

۵-۱-۴ ترسیم نمودارهای تو در تو

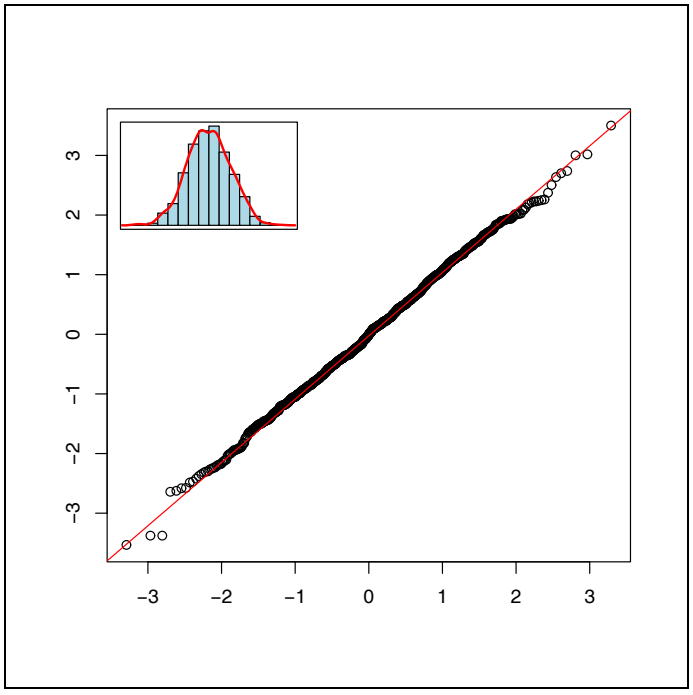
در این قسمت می‌توان یک نمودار را در داخل نموداری دیگر درج نمود. کُدهای آن به‌صورت زیر است.

```
# Nested graphics
library(e1071)
n <- 1000
x <- rnorm(n)
qqnorm(x, main=paste("kurtosis =", round(kurtosis(x), digits=2),
"(gaussian)"))
qqline(x, col="red")
op <- par(fig=c(.02,.5,.5,.98), new=TRUE)
hist(x, probability=T,
col="light blue", xlab="", ylab="", main="", axes=F)
lines(density(x), col="red", lwd=2)
box()
```

با اجرای کُدهای فوق شکل ۵-۴ حاصل می‌گردد. ضمناً برای رسم این نمودار از بسته e1071 استفاده شده است. زیرا تابع کشیدگی kurtosis() در خود زبان R موجود نیست.



شکل ۵-۳: نمایش رگرسیون و توابع حاشیه‌ای



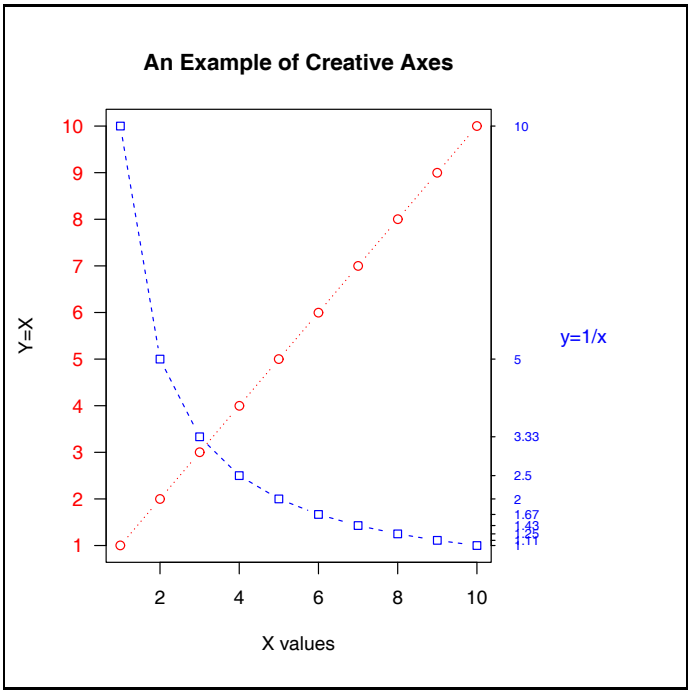
شکل ۵-۴: نمایش نمودارهای تو در تو

۵-۱-۵ ترسیم دو محور عرض‌ها در یک نمودار

در پاره‌ای از حالات به رسم دو محور y در یک نمودار نیاز پیدا می‌شود. برای این مورد به مثال زیر توجه کنید.

```
# specify the data
x <- c(1:10); y <- x; z <- 10/x
# create extra margin room on the right for an axis
par(mar=c(5, 4, 4, 8) + 0.1)
# plot x vs. y
plot(x, y,type="b", pch=21, col="red",
yaxt="n", lty=3, xlab="", ylab="")
# add x vs. 1/x
lines(x, z, type="b", pch=22, col="blue", lty=2)
# draw an axis on the left
axis(2, at=x,labels=x, col.axis="red", las=2)
# draw an axis on the right, with smaller text and ticks
axis(4, at=z,labels=round(z,digits=2),
col.axis="blue", las=2, cex.axis=0.7, tck=-.01)
# add a title for the right axis
mtext("y=1/x", side=4, line=3, cex.lab=1,las=2, col="blue")
# add a main title and bottom and left axis labels
title("An Example of Creative Axes", xlab="X values", ylab="Y=X")
```

با اجرای کُدهای فوق شکل ۵-۵ به‌دست می‌آید.



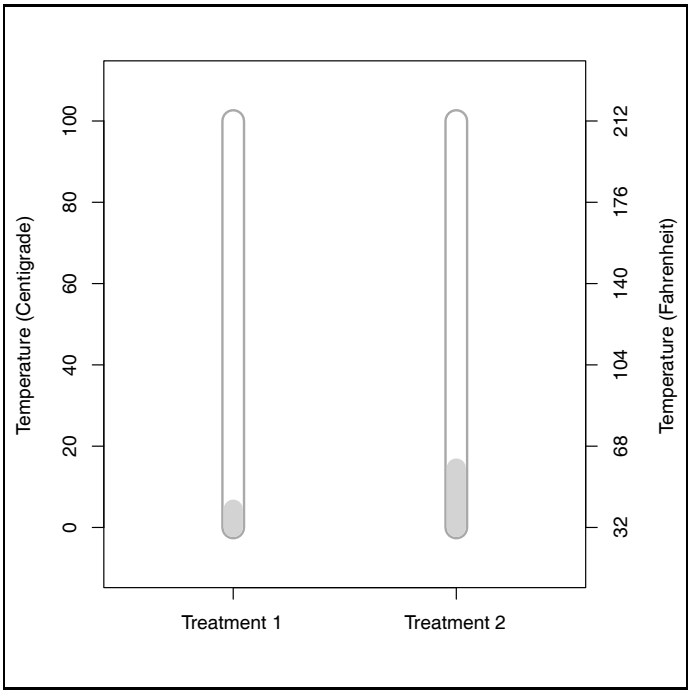
شکل ۵-۵: نمایش یک نمودار با دو محور عرض‌ها

۵-۱-۶ ترسیم دو درجه حرارت در یک نمودار

می‌توان نموداری داشت که دارای دو محور عرض‌ها باشد، یکی بر سانتی گراد و دیگری بر حسب فارنهایت هستند. برای این مورد به مثال زیر توجه کنید.

```
x <-1:2
y <- runif(2, 0, 100)
par(mar=c(4, 4, 2, 4))
plot(x, y, type="n", xlim=c(0.5, 2.5), ylim=c(-10, 110),
axes=FALSE, ann=FALSE)
axis(2, at=seq(0, 100, 20))
mtext("Temperature (Centigrade)", side=2, line=3)
axis(1, at=1:2, labels=c("Treatment 1", "Treatment 2"))
axis(4, at=seq(0, 100, 20), labels=seq(0, 100, 20)*9/5 + 32)
mtext("Temperature (Fahrenheit)", side=4, line=3)
box()
segments(x, 0, x, 100, lwd=20, col="dark grey")
segments(x, 0, x, 100, lwd=16, col="white")
segments(x, 0, x, y, lwd=16, col="light grey")
```

با اجرای کُدهای فوق شکل ۵-۶ به‌دست می‌آید.



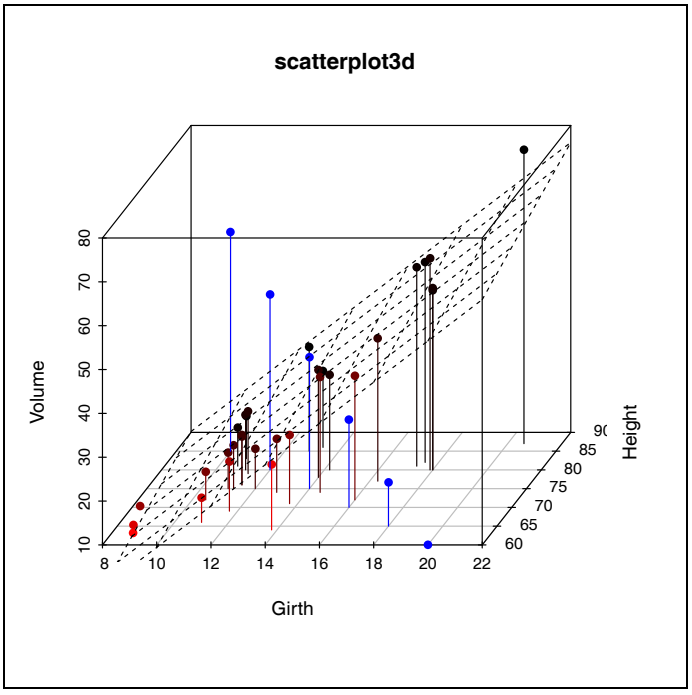
شکل ۵-۶: نمایش یک نمودار با دو محور دما

۷-۱-۵ ترسیم صفحه رگرسیون

همان‌طورکه قبلاً مشاهده شد رابطه رگرسیون بین y, x می‌تواند به‌صورت خط مستقیم باشد. اکنون اگر معادله رگرسیون به‌صورت $y = a_1x_1 + a_2x_2 + b$ باشد، نمایش آن نموداری به شکل یک صفحه است. اکنون در کُدهای زیر برای رگرسیون در مورد داده trees در R ترسیم صفحه انجام شده است. در این ترسیم رگرسیون Volume بر حسب متغیرهای Girth و Height انجام شده است.

```
library(scatterplot3d)
data(trees)
s3d <- scatterplot3d(trees, type="h", highlight.3d=TRUE,
angle=55, scale.y=0.7, pch=16, main="scatterplot3d")
# Now adding some points to the "scatterplot3d"
s3d$points3d(seq(10,20,2), seq(85,60,-5), seq(60,10,-10),
col="blue", type="h", pch=16)
# Now adding a regression plane to the "scatterplot3d"
attach(trees)
my.lm <- lm(Volume ~ Girth + Height)
s3d$plane3d(my.lm)
```

با اجرای کُدهای فوق شکل ۵-۷ به‌دست می‌آید.



شکل ۵-۷: نمایش صفحه رگرسیون

مراجع

[1] **Crawley, Micheal J., (2007)** *The R Book,* John Wiley & Sons Ltd, 942p.

[2] **Longhow Lam, (2008)** *An introduction to R,* Business & Decision Amsterdam, 212p.

[3] **Maindonald J. H., (2008)** *Using R for Data Analysis and Graphics Introduction, Code and Commentary,* Centre for Mathematics and Its Applications, Australian National University., 96p.

[4] **Owen Jones, Robert Maillardet, and Andrew Robinson, (2009)** *IntroductIon to Scientific Programming and Simulation using r,* Chapman & Hall/CRC, 448p.

[5] **Paradis E., (2005)** *R for Beginners,* Institut des Sciences de l' Evolution Universite Montpellier II France, 76p.

[6] **Rossiter D. G., (2007)** *Introduction to the R Project for Statistical Computing for use at ITC,* International Institute for Geo-information Science & Earth Observation Enschede (NL), 143p.

[7] **Sarkar Deepayan, (2008)** *Multivariate Data Visualization with R,* Springer, 265p.

[8] **Seefeld, K., Linder, E., (2007)** *Statistics Using R with Biological Examples,* University of New Hampshire, Durham, NH Department of Mathematics & Statistics, 325p.

[9] **Scott A Theresa, (?)** *An Introduction to the Fundamentals & Functionality of the R Programming Language,* Vanderbilt University, Department of Biostatistics, 99p.

[10] **Soetaert, Karline, (2008)** *Using R for scientific computing ,* Center for Estuarine and Marine Ecology Netherlands Institute of Ecology, 46p.

[11] **Spector Phil, (2008)** *Data Manipulation with R,* Springer, 152p.

[12] **Verzani John** *simpleR - Using R for Introductory Statistics,* 114p.