معرفی نرم افزار *S-PLUS*

این نرم افزار یکی از نرم افزارهای پیشرفته دهه اخیر می باشد که با سرعت قابل ملاحظه ای پیشرفت داشته است و نسخه های مختلف آن هر چند وقت یکبار به جهان عرضه می شود. این نرم افزار با برخورداری از کارایی وسیع در نظریه های مختلف آمار در سطوح مقدماتی و پیشرفته خصوصاً در سال های اخیر به عنوان ابزاری مهم در عمده تحلیل داده ها و مدل های آماری و استخراج نتایج توسط پژوهشگران جهان مورد استفاده قرار می گیرد. در واقع نرم افزار *S* توسط گروه (*Bell Labs S*) متعلق به شرکت *AT&T* تهیه شده که نسخه کامل تر شده آن معروف به *S+* می باشد. این نرم افزار علاوه بر داشتن توابع آماری و ریاضی، یک محیط برنامه نویسی توانا در اختیار کاربران قرار می دهد. ویژگی دیگر این نرم افزار ارائه توابعی برای انجام روش های جدید آماری از جمله روشهای استوار است. *S+* با بیش از ۴۲۰۰ تابع پیش ساخته و محیط فعال برای ورود داده ها و تحلیل اطلاعات و همچنین محیط گرافیکی قوی و گستردگی نمودارها، که در اختیار هر کدام از موضوعات مختلف آماری قرار داده است، به درک و فهم بیشتر و راحت تر اطلاعات و موضوعات کمک شایانی می نماید. همچنین زبان برنامه نویسی آن به گونه ای است که توابع و برنامه های پیشرفته را می تواند با حجم کمی از دستورات به مرحله انجام برساند، بدون اینکه محقق به زبان برنامه نویسی دیگری احتیاج داشته باشد. این نرم افزار قابلیت های بسیاری در زمینه مدل های قدیمی، تکنیک های مدرن و نظریه های نوین آمار از قبیل ناپارامتری، رگرسیون استوار، موجک ها و ... دارد.

این نرم افزار محصول سال‌های ۱۹۸۸- ۱۹۹۹ است که از طرف شرکت *Mathsoft* تدوین شده است و ساختارش به گونه‌ای است که با نرم‌افزارهای *Office* و *Spss* تبادل اطلاعاتی خوبی انجام می‌دهد، یعنی یک آیکون در آن نرم افزارها قرار می‌دهد که می‌توان در حال استفاده از آن نرم افزارها با *S-Plus* تبادل اطلاعاتی انجام داد. نویسندگان این نرم افزار عبارتند از

Richard A. Becker, John M. Chambers, Allan R. Wilks, William S. Cleveland, Trevor Hastie.

امکانات و ویژگی های نرم افزار

قابلیت نمایش و آنالیز مجموعه بزرگی از داده ها، داشتن محیط گرافیکی قوی و همچنین مدلهای ژئوگرافیکی با توانایی رسم برخی نقشه های *GIS* ، داشتن پنجره *help* کامل و دقیق، انجام انواع تست‌های ناپارامتریک، انجام آنالیز واریانس یک‌طرفه و چندطرفه، انجام رگرسیون خطی و غیرخطی، رسم نمودارهای کنترل کمی و کیفی، قابلیت تجزیه و تحلیل سری‌های زمانی، رسم نمودارهای دو بعدی و سه‌بعدی در انواع مختلف بالغ بر بیش از ۱۰۰ نوع نمودار، امکان برنامه نویسی در محیط نرم‌افزار، داشتن سلسله روش های کامل برای تحلیل سری های زمانی، ارائه چندین روش جهت بررسی داده های گمشده و همچنین ارائه روش های قوی آماری جهت قابل اطمینان تر ساختن برآوردها

ساختار داده ها

برای انجام محاسبات ساده ریاضی می توان از نرم افزار استفاده کرد.

```
> 2 + 2 * 2
[1] 6
> (7 + 5 - 8^2) / (19 * 2)
[1] -1.368421
```

برای ساختن یک بردار روش های متنوعی وجود دارد که در زیر تعدادی از آنها معرفی می شوند.

```
> x <- c(7,8,5,3,8,6,6,9,5,4,1,5)
> x
 [1] 7 8 5 3 8 6 6 9 5 4 1 5
> y <- scan()
1: 4 5 6 5 4 3 2 3 4 5 6 7
13:
```

نکته: فرم متداول برای انتساب نماد <- است و جایگزین های آن دو نماد = و _ می باشند.

```
> x = c(4,3,2,1,2,3,4,4,5,4,2,4)
> x _ c(5,4,3,2,3,4,3,2,4,3,1,4)
```

برای تعریف اعداد متوالی از نماد : می توان استفاده کرد.

```
> z <- 10:20
> z
 [1] 10 11 12 13 14 15 16 17 18 19 20
```

برای مشخص کردن و اشاره به عددی خاص در یک بردار از تعاریف زیر می توان استفاده کرد.

```
> z[4]
 [1] 13
> z[z > 13]
 [1] 14 15 16 17 18 19 20
> z[c(2,3,4)]
 [1] 11 12 13
> z[-c(2,3,4)]
 [1] 10 14 15 16 17 18 19 20

> x <- c(1.9, 3.0, 4.1, 2.6, 3.6, 2.3, 2.8, 3.2, 6.6, 7.6, 7.4, 1.0)
> x[12:1]
[1] 1.0 7.4 7.6 6.6 3.2 2.8 2.3 3.6 2.6 4.1 3.0 1.9
> x[-(3:5)]
[1] 1.9 3.0 2.3 2.8 3.2 6.6 7.6 7.4 1.0
> x[-13]
[1] 1.9 3.0 4.1 2.6 3.6 2.3 2.8 3.2 6.6 7.6 7.4 1.0
> x[x > 2]
[1] 3.0 4.1 2.6 3.6 2.3 2.8 3.2 6.6 7.6 7.4
```

برای تعریف دنباله ای از اعداد تابع *seq* استفاده می شود.

```
> seq(to = 17)          # same as 1:17
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
> seq(from = 1, to = 25, by = 2)
 [1] 1 3 5 7 9 11 13 15 17 19 21 23 25
> seq(from = 0, to = 1, length = 11)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

برای تعریف ترکیبات تکراری از اعداد تابع *rep* استفاده می شود.

```
> rep(5, times = 10)
 [1] 5 5 5 5 5 5 5 5 5 5
> rep(c(3,6,5), times = 6)
 [1] 3 6 5 3 6 5 3 6 5 3 6 5 3 6 5 3 6 5
> rep(c(2,3,4), times = rep(3,3))
 [1] 2 2 2 3 3 3 4 4 4
```

نکته: مقادیر گمشده با *NA* (*Not Available*) نشان داده می شوند.

```
> x <- c(2, 3, 4, NA, 3, 2, 1, NA)
```

برای حذف مقادیر گمشده یک متغیر از دستورات زیر می توان استفاده کرد.

```
> x <- x[!is.na(x)]
> x1 <- na.omit(x)
> x2 <- na.exclude(x)
```

برای استفاده از راهنمای دستورات و توابع از یکی از دو شکل زیر می توان استفاده کرد.

```
> ? cor.test
> help(t.test)
```

نکته: تعدادی از مجموعه داده های موجود در نرم افزار عبارتند از

kyphosis, longley.x, ethanol, auto.stats, geyser, claims, solder, air, swiss.x, switzerland
fuel.frame, wafer, sunspots, freeny.y,lynx, ship, co2, rain.nyc2

برای مشاهده یک مجموعه داده با تایپ نام می توان جزئیات داده ها را مشاهده کرد.

```
> ship
```

معرفی عملگرهای منطقی و مقایسه ای و ارائه مثال هایی برای آنها

عملگر	توصیف
== , !=	*Equal to , Not equal to*
< , >	*Greater (less) than*
<= , >=	*Greater (less) than or equal*
& , |	*Vectorized And , Or*
&& , ||	*Control And , Or*
!	*Not*

در نرم افزار دو شکل (برداری و کنترل) از توابع *AND* و *OR* وجود دارد. عملگرهای برداری توابع *AND* و *OR* را عضو به عضو ارزیابی می کنند و برداری شامل مقادیر *T* و *F* به خروجی می فرستند. در صورتی که عملگرهای کنترل به جای برداری از مقادیر منطقی فقط یک مقدار منطقی بر می گردانند. بهمین دلیل معمولاً عملگرهای برداری هستند که کاربرد بیشتری دارند.

مثال هایی برای استفاده از عملگرهای منطقی در انجام مقایسات

```
> z.vector <- 1:10
> z.vector > 5
[1] F F F F F T T T T T

> index <- (z.vector > 5)
> newz.vector <- z.vector[index]
> newz.vector
[1] 6 7 8 9 10

> z.vector == 5
[1] F F F F T F F F F F
> (z.vector > 5)&(z.vector < 8)
[1] F F F F F T T F F F
> (z.vector < 5)|(z.vector > 8)
[1] T T T T F F F F T T

> all(z.vector < 5)
[1] F
> any(z.vector < 5)
[1] T
> all(z.vector < 5) && any(z.vector < 5)
[1] F
> all(z.vector < 5) || any(z.vector < 5)
[1] T
```

و مثالی دیگر

```
> x <- c(1.9, 3.0, 4.1, 2.6, 3.6, 2.3, 2.8, 3.2, 6.6, 7.6, 7.4, 1.0)
> x[x > 2 & x < 4]
[1] 3.0 2.6 3.6 2.3 2.8 3.2
> x < 2 | x > 4
```

```
[1] T F T F F F F F T T T T
> x > 2 & x < 4
[1] F T F T T T T T F F F F
> all(x < 2) || any(x > 4)
[1] T
> all(x < 2) && any(x > 4)
[1] F
```

در برنامه زیر تا وقتی که حاصل عبارت $i < 6 \, , \, i + j < 9$ برابر T است حلقه *while* عمل می کند.

```
> i <- 3
> j <- 3
> while(i < 6 && i + j <= 9){
+     j <- j - 1
+     i <- i + 1
+     cat(i, j, "\n")
+   }
4 2
5 1
6 0
```

در برنامه زیر اگر مقدار a خارج بازه صفر و یک باشد پیغام هشدار چاپ می شود و در غیر اینصورت مربع مقدار a محاسبه می شود.

```
> a <- 0.5
> if(a < 0 || a > 1) stop("a must be between 0 and 1")
> square <- a ^ 2
> square
[1] 0.25
```

تابعی برای محاسبه موجک هار (*Haar Wavelet*) به شکل زیر می باشد.

$$\psi^{(H)}(u) = \begin{cases} -۱/\sqrt{۲} & -۱ < u \leq ۰ \\ ۱/\sqrt{۲} & ۰ < u \leq ۱ \\ ۰ & o.w \end{cases}$$

```
> haar <- function(x){
  y <- x * 0
  for(i in 1:length(y)){
  if (x[i] < 0 && x[i] > -1) y[i] = -1/sqrt(2)
  else if (x[i] > 0 && x[i] < 1) y[i] = 1/sqrt(2)
   }
  y
  }
```

برای ایجاد یک ماتریس از تابع *matrix* به شکل زیر می توان استفاده کرد. توابع مرتبط با ماتریس به همراه مثال هایی در ادامه معرفی می شوند.

```
> matrix(1:4, ncol = 2)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> matrix(1:4, ncol = 2, byrow = T)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
>
> A <- matrix(1:9, nrow = 3)
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
```

```
[3,]    3    6    9
```

از توابع *cbind* و *rbind* نیز برای ایجاد یک ماتریس می توان استفاده کرد.

```
> cbind(c(12,15,13), c(17,19,13), c(11, 16,14))
     [,1] [,2] [,3]
[1,]   12   17   11
[2,]   15   19   16
[3,]   13   13   14

> rbind(c(12,15,13), c(17,19,13), c(11, 16,14))
     [,1] [,2] [,3]
[1,]   12   15   13
[2,]   17   19   13
[3,]   11   16   14

> B <- matrix(c(5,3,6,3,1,2,8,6,5), nrow=3, ncol=3)
> B
     [,1] [,2] [,3]
[1,]    5    3    8
[2,]    3    1    6
[3,]    6    2    5
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

در ادامه نحوه محاسبه مجموع، حاصلضرب، بعد و ترانهاده یک ماتریس معرفی شده است.

```
> A + B
     [,1] [,2] [,3]
[1,]    6    7   15
[2,]    5    6   14
[3,]    9    8   14
> A %*% B
     [,1] [,2] [,3]
[1,]   59   21   67
[2,]   73   27   86
[3,]   87   33  105
> dim(A)
[1] 3 3
> t(B)
     [,1] [,2] [,3]
[1,]    5    3    6
[2,]    3    1    2
[3,]    8    6    5
```

چارچوب اطلاعاتی تعمیمی از یک ماتریس است و بوسیله آن می توان انواع مختلفی از داده ها را با هم ترکیب کرده و برای تجزیه و تحلیل مورد استفاده قرار داد.

```
> x1 <- rpois(5, 2); x2 <- rpois(5, 3); x3 <- rpois(5, 4)
> xyz <- data.frame(x1, x2, x3)
> xyz
  x1 x2 x3
1  1  7  4
2  3  2  1
3  2  2  3
4  2  2  3
5  3  4  4
```

برای ذخیره و فراخوانی داده ها از یک فایل از توابع *write.table* **و** *read.table* **استفاده می شود.**

```
> x <- c(4,6,6,8,7,6,4,6,7,3,1,5,7,4,6,3)
> write.table(x, file = "D:\\data.txt")
> y <- read.table("D:\\data.txt", header = T)
> y
```

برای طبقه بندی (کد گذاری) داده های پیوسته و گسسته از توابع *cut* **و** *category* **استفاده می شود.**

```
> x <- c(7,8,5,3,8,6,6,9,5,4,1,5)
> cut(x, breaks = c(0,3,9))
 [1] 2 2 2 1 2 2 2 2 2 2 1 2
attr(, "levels"):
 [1] "0+ thru 3" "3+ thru 9"

> cuts <- cut(rpois(100,6), breaks = c(0,4,8,12,16))
> table(cuts)
  0+ thru  4  4+ thru  8  8+ thru 12 12+ thru 16
         26          61          11           2

> category(c('A','B','A','AB','O','A','B','AB','O'))
 [1] 1 3 1 2 4 1 3 2 4
attr(, "levels"):
 [1] "A"  "AB"  "B"  "O"
```

توابع مربوط به محاسبات ریاضی

در ادامه در قالب مثال هایی تعدادی از توابع مفید و پرکاربرد ریاضی و آماری معرفی می شوند.

```
> x
 [1] 7 8 5 3 8 6 6 9 5 4 1 5
```

نمادهای *%%, %/%* **برای محاسبه باقیمانده و مقسوم علیه استفاده می شوند.**

```
> x %% 2
 [1] 1 0 1 1 0 0 0 1 1 0 1 1
> x %/% 2
 [1] 3 4 2 1 4 3 3 4 2 2 0 2
```

توابع محاسبه ترکیب، فاکتوریل، گاما، لگاریتم گاما عبارتند از

```
> choose(5,2)
 [1] 10
> factorial(5)
 [1] 120
> choose.multinomial(6, c(3,1,2))
[1] 60
> gamma(4)
[1] 6
> lgamma(4)
[1] 1.791759
> digamma(4)
[1] 1.256118
> trigamma(4)
[1] 0.283823
```

نکته: توابع گاما، گامای ناقص، بتا، بتای ناقص، دی گاما و تری گاما به ترتیب به صورت زیر تعریف می شوند.

$$\Gamma(a) = \int_{0}^{+\infty} x^{a-1} e^{-x} dx = (a-1)\Gamma(a-1) \qquad \forall\ a > 0$$

$$\gamma(a,x) = \int_{0}^{x} y^{a-1} e^{-y} dy \quad , \quad \Gamma(a,x) = \int_{x}^{+\infty} y^{a-1} e^{-y} dy \qquad \forall\ a,x > 0$$

$$\beta(a,b) = \int_{0}^{1} x^{a-1} (1-x)^{b-1} dx = \Gamma(a)\Gamma(b)/\Gamma(a+b) \qquad \forall\ a,b > 0$$

$$\beta(a,b,u) = \int_{0}^{u} x^{a-1} (1-x)^{b-1} dx \qquad \forall\ 0 \le u \le 1$$

$$\psi(x) = \frac{d}{dx}\ln\Gamma(x) = \Gamma'(x)/\Gamma(x) \qquad \forall\ x > ۰$$

$$\psi_۱(x) = \frac{d}{dx}\psi(x) = \frac{\Gamma''(x)}{\Gamma(x)} - \frac{\Gamma'(x)^۲}{\Gamma(x)^۲} \qquad \forall\ x > ۰$$

توابع مربوط به گرد کردن عبارتند از

```
> r <- c(3.27,4.21,-5.55,-7.32,8.11,-4.78)
> ceiling(r)
 [1]  4  5 -5 -7  9 -4
> floor(r)
 [1]  3  4 -6 -8  8 -5
> trunc(r)
 [1]  3  4 -5 -7  8 -4
> round(r)
 [1]  3  4 -6 -7  8 -5
> round(r, digits = 1)
 [1]  3.3  4.2 -5.6 -7.3  8.1 -4.8
```

توابع مربوط به محاسبه قدرمطلق و علامت به ترتیب *abs* **و** *sign* **می باشند.**

```
> abs(r)
 [1] 3.27 4.21 5.55 7.32 8.11 4.78
> sign(r)
 [1]  1  1 -1 -1  1 -1
```

توابع مربوط به رتبه گذاری و مرتب سازی داده ها عبارتند از

```
> sort(x)
 [1] 1 3 4 5 5 5 6 6 7 8 8 9
> rank(x)
 [1] 9.0  10.5  5.0  2.0  10.5  7.5  7.5  12.0  5.0  3.0  1.0  5.0
> rev(x)
 [1] 5 1 4 5 9 6 6 8 3 5 8 7
> rev(sort(x))
 [1] 9 8 8 7 6 6 5 5 5 4 3 1
> rev(rank(x))
 [1] 5.0  1.0  3.0  5.0  12.0  7.5  7.5  10.5  2.0  5.0  10.5  9.0
```

تابع *order* **موقعیت هر عدد را در بردار مشخص می کند. به عنوان مثال در بردار زیر کوچکترین عدد (۲) در موقعیت ۳ ، عدد بزرگتر بعدی (۳) در موقعیت ۵ و ... قرار دارند.**

```
> order(c(8,4,2,6,3))
 [1] 3 5 2 4 1
> order(1:5)
 [1] 1 2 3 4 5
> order(5:10)
 [1] 1 2 3 4 5 6
```

توابع جمع تجمعی، مینیمم تجمعی، ماکزیمم تجمعی و ضرب تجمعی عبارتند از

```
> cumsum(1:10)
 [1]  1  3  6  10  15  21  28  36  45  55
> cummin(x)
 [1] 7 7 5 3 3 3 3 3 3 3 1 1
> cummax(x)
 [1] 7 8 8 8 8 8 8 9 9 9 9 9
> cumprod(1:7)
 [1]  1  2  6  24  120  720  5040
```

توابع مربوط به محاسبه انواع لگاریتم، تابع نمایی و تبدیل فوریه عبارتند از

```
> log(c(7,8,5,3))
 [1] 1.945910 2.079442 1.609438 1.098612
```

```
> log10(c(7,8,5,3))
 [1] 0.8450980 0.9030900 0.6989700 0.4771213
> logb(5, base = 3)
 [1] 1.464974
> exp(c(7,8,5,3))
 [1] 1096.633158  2980.957987  148.413159  20.085537
```

محاسبه انتگرال معین

```
> integrate(sin, lower=0, upper=pi, subdivisions=200)$integral
 [1] 2
> integrate(function(x) exp(x), lower=0, upper=3)[1:2]
$integral:
[1] 19.08554
$abs.error:
[1] 2.11892e-013
> integrate(function(x) x^3-3*x^2+2*x, lower=0, upper=4)[1]
$integral:
[1] 16
> integrate(function(x) exp(-x^2+x), lower=0, upper=Inf)[1]
$integral:
[1] 1.730234
```

توابع مربوط به آمار توصیفی

توابع ساده مربوط به محاسبه شاخص های مختلف آمار توصیفی عبارتند از

```
> sum(x)
 [1] 67
> prod(x)
 [1] 217728000
> min(x)
 [1] 1
> max(x)
 [1] 9
> mean(x)
 [1] 5.583333
> mean(x, trim = 0.1)
 [1] 5.7
> median(x)
 [1] 5.5
> var(x)
 [1] 5.174242
> stdev(x)
 [1] 2.274696
```

اگر بردار شامل مقادیر گمشده باشد می بایست از شناسه $na.rm=T$ **جهت انجام محاسبات استفاده کرد.**

```
> y <- c(4,3,7,NA,2,1,NA,6,4)
> mean(y)
 [1] NA
> mean(y, na.rm = T)
 [1] 3.857143
> median(y, na.rm = T)
 [1] 4
```

تابع *range* **مقادیر ماکزیمم و مینیمم بردار داده ها را مشخص می کند.**

```
> range(x)
 [1] 1 9
> diff(range(x))
 [1] 8
> mad(x, center = mean(x), constant = 1.4826)
```

```
 [1] 2.2239
> mad(x, center = median(x), constant = 1.4826)
 [1] 2.2239
> 1.4826 * median(abs(x - mean(x)))
 [1] 2.2239
> 1.4826 * median(abs(x - median(x)))
 [1] 2.2239
```

تابعی برای ارائه آمار توصیفی خلاصه شده در ۶ شاخص عبارتست از

```
> summary(x)
 Min. 1st Qu. Median   Mean 3rd Qu. Max.
    1    4.75    5.5  5.583    7.25    9
> Range <- summary(x)[6] - summary(x)[1]
> Range
    8
> Range <- range(x)[2] - range(x)[1]
> Range
    8
> IQR <- summary(x)[5] - summary(x)[2]
> IQR
     2.5
> TM <- (summary(x)[2] + 2 * summary(x)[3] + summary(x)[5]) / 4
> TM
 5.33325
```

برای محاسبه چندک های مختلف داده ها از تابع *quantile* **استفاده می شود.**

```
> quantile(x)
   0%  25%  50%  75% 100%
    1 4.75  5.5 7.25    9
> quantile(x, probs = c(0.22, 0.38, 0.61, 0.73, 0.94))
   22% 38% 61%  73%  94%
  4.42   5    6 7.03 8.34
> quantile(x, probs = seq(0, 1, 0.1))
   0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
    1  3.1  4.2    5    5  5.5    6  6.7  7.8    8    9
> iqr <- quantile(x)[4] - quantile(x)[2]
> iqr
  2.5
> range <- quantile(x)[5] - quantile(x)[1]
> range
    8
```

توابع مفید آماری برای شبیه سازی، محاسبه احتمال تجمعی، چندک و چگالی از توزیع های مختلف

```
> rnorm(5, 1, 2)
[1] -2.5298360 -2.8502458 4.3493681 -0.7775381 -1.0524232
> pnorm(1.64, 0, 1)
[1] 0.9494974
> qnorm(0.975, 0, 1)
[1] 1.959964
> dnorm(0, 0, 1)
[1] 0.3989423

> rexp(5, rate = 3)
[1] 0.28711865 0.17706108 0.30943034 0.75169239 0.09015125
> pexp(2, rate = 3)
[1] 0.9975212
> qexp(0.95, rate = 3)
[1] 0.9985774
> dexp(1, rate = 3)
```

```
[1] 0.1493612
```

دو نمونه فوق برای توزیع های نرمال و نمایی هستند برای سایر توزیع ها کافیست توابع زیر بـه همراه حروف *d,q,p,r* **جهت شبیه سازی، محاسبه احتمال تجمعی، چندک و چگالی استفاده شوند.**

beta, binom, nbinom, pois, geom, hyper, wilcox, unif, T, F, cauchy, chisq, gamma, lnorm, logis, weibull

جدولی برای توزیع های مختلف، پارامترهای آنها و مقادیر پیش فرض

Code	*Distribution*	*Parameters*	*Defaults*
beta	*beta*	*shape1, shape2*	*--, --*
binom	*binomial*	*size, prob*	*--, --*
Cauchy	*Cauchy*	*location, scale*	*0, 1*
chisq	*chi squared*	*Df*	*-*
exp	*exponential*	*Rate*	*1*
f	*F*	*df1, df2*	*--, --*
gamma	*gamma*	*shape, rate*	*--, 1*
geom	*geometric*	*Prob*	*--*
hyper	*hyper geometric*	*m, n, k*	*--, --, --*
lnorm	*lognormal*	*meanlog, sdlog*	*0, 1*
logis	*logistic*	*location, scale*	*0, 1*
nbinom	*negative binomial*	*size, prob*	*--, --*
norm	*Normal*	*mean, sd*	*0, 1*
pois	*Poisson*	*Lambda*	*1*
stab	*Stable*	*index, skew*	*--, 0*
t	*Student's t*	*Df*	*--*
unif	*uniform*	*min, max*	*0, 1*
weibull	*Weibull*	*shape, scale*	*--, 1*
wilcoxon	*Wilcoxon*	*m, n*	*--, --*

برنامه ای برای شبیه سازی ۳۰ مرتبه پرتاب یک سکه

```
> r = rbinom(30, 1, 0.5)
> r
 [1] 1 1 1 0 1 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 0 0 0 1 1 0 1 0 0 0
> mean(r)
[1] 0.5333333
```

برنامه ای برای محاسبه احتمال قبولی یک دانشجو در آزمونی با ۲۰ سوال ۴ گزینه ای که پاسخ به سوالات به طور تصادفی داده می شود، بصورت زیر است.

```
> sum = 0
> for(i in 10:20) sum = sum + (pbinom(i, 20, 0.25)-pbinom(i-1, 20, 0.25))
> sum
[1] 0.014
```

که در آن $X_i \sim b(۱,۰/۲۵)$ **بنابراین** $\sum_{i=۱}^{۲۰} X_i \sim b(۲۰,۰/۲۵)$ **و احتمال قبولی** $p(\sum_{i=۱}^{۲۰} X_i \geq ۱۰)$ **خواهد بود.**

برنامه ای برای شبیه سازی ۳۰ مرتبه پرتاب یک تاس

```
> y = runif(30, 0, 6)
> y1 = cut(y, breaks = c(0,1,2,3,4,5,6))
> y1
[1] 5 1 1 5 3 1 3 2 1 5 3 4 4 5 5 2 5 1 3 6 5 6 2 5 3 6 6 3 6 3
> table(y1)
 0+ thru 1 1+ thru 2 2+ thru 3 3+ thru 4 4+ thru 5 5+ thru 6
         5         3         7         2         8         5
> table(y1)/30
```

```
 0+ thru 1 1+ thru 2 2+ thru 3  3+ thru 4 4+ thru 5 5+ thru 6
 0.1666667       0.1 0.2333333 0.06666667 0.2666667 0.1666667
```

برنامه ای برای محاسبه احتمال مشاهده صفر الی چهل توپ قرمز در نمونه ای شامل ۴۰ توپ از ظرفی که در آن تعداد مساوی توپ سفید و قرمز وجود دارد.

```
> numballs <- 40
> p <- 0.5
> n <- c(0:numballs)
> probs <- c()
> for (k in n) probs[k] <- pbinom(numballs, p, k)
> round(probs, digits=3)
[1]  0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.001
[12] 0.003 0.008 0.019 0.040 0.077 0.134 0.215 0.318 0.437 0.563 0.682
[23] 0.785 0.866 0.923 0.960 0.981 0.992 0.997 0.999 1.000 1.000 1.000
[34] 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
```

تعریف توابع جدید

در این قسمت با ارائه مثال هایی ساده زمینه آشنایی بیشتر با نحوه نوشتن توابع جدید را فراهم می کنیم.

تابعی که مقدار ۱ + ۲x + x^2 $f(x) = x^2 + 2x + 1$ را محاسبه می کند.

```
> equation <- function(x) {x^2 + 2*x + 1}
> equation(3)
[1] 16
```

تابعی که توان k ام عددی ورودی را محاسبه می کند.

```
> power.x <- function(x, k = 2){ x ^ k }
> power.x(5,2)
[1] 25
```

تابعی برای محاسبه انحراف معیار

```
> std.dev <- function(x) sqrt(var(x))
> std.dev(x)
```

تابعی برای محاسبه مجموع دو عدد ورودی

```
> add = function(a,b){
   result = a + b
   return(result)
  }
> add(5,8)
[1] 13
```

تابعی برای محاسبه درجه ۲ و ۳ عدد یا بردار ورودی

```
> sqacu_function(x){
        res1_x^2
        res2_x^3
        return(list("square"=res1, "cube"=res2))
  }
> sqacu(2)
 $square:
[1] 4
 $cube:
[1] 8
> sqacu(2)$square
[1] 4
```

تابعی برای محاسبه مقادیر استاندارد شده

```
> standardize_function(x){
        m_mean(x)
        std_sqrt(var(x))
```

```
        result_(x - m)/std
        return(result)
  }
> standardize(c(3,6,5,7,6,3,7,5,2))
```

تابعی که مقدار تابع خطا[1] را محاسبه می کند. $erf(x) = \frac{۲}{\sqrt{\pi}}\int_{.}^{x} e^{-t^{۲}}\, dt$

```
> myerf <- function(x)  2 * pnorm(x * sqrt(2)) - 1
> myerf(1)
[1] 0.8427008
> myerf(1.96)
[1] 0.9944263
```

نکته: برای ویرایش یک تابع نوشته شده از تابع *fix* **به شکل** *fix(erf)* **استفاده می شود.**

نکته: برای ارائه نتایج از ۴ تابع *return, print, cat, list* **استفاده می شود.**

ساختار کلی حلقه *for* **به شکل زیر است.**

```
for(var in seq) expr
```

```
> for(i in 1:5) print(1:i)
[1] 1
[1] 1 2
[1] 1 2 3
[1] 1 2 3 4
[1] 1 2 3 4 5

> for(i in 1:5){
+  print(i*i)
+ }
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25

> for(n in c(2,5,10,20,50)){
    x <- rnorm(n)
    cat(n, ":", sum(x ^ 2), "\n")
  }
2  : 1.71727379371661
5  : 2.68333093515954
10 : 9.72417592478934
20 : 21.0137090893203
50 : 66.5794803919901
```

نکته: استفاده از *"\n"* **منجر به ارائه نتایج بصورت زیر هم می شود که با حذف آن نتایج بصورت کنار هم و بصورت سطری ارائه می شوند.**

```
> for (n in 0:4) print(choose(n, k = 0:n))
[1] 1
[1] 1 1
[1] 1 2 1
[1] 1 3 3 1
[1] 1 4 6 4 1
```

از تابع نیز برای ارائه نتایج در خروجی می توان استفاده کرد.

```
> my.power <- function(x){
```

[1] *Error function*

```
      square <- x ^ 2
      cubic <- x ^ 3
      c(square = square, cubic = cubic)
  }
> my.power(x = 5)
 square  cubic
     25    125
```

ساختار کلی شرط های *if* **و** *ifelse* **به شکل زیر است.**

```
if(cond) expr
if(cond) cons.expr  else  alt.expr
ifelse(test, yes, no)

> g <- rnorm(10)
> ifelse(g > 0, 1, ifelse(g < 0, -1, 0))
 [1] -1 -1  1 -1  1  1 -1 -1 -1 -1
```

ساختار کلی حلقه های *while* **و** *repeat* **به شکل زیر است.**

```
while(cond) expr
repeat expr
break
next
```

در برنامه های زیر تا وقتی i **کوچکتر مساوی ۵ باشد محاسبه و چاپ نتایج ادامه می یابد.**

```
> i = 1
> while(i <= 5){
+    print(i)
+    i = i + 1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5

> i = 1
> repeat{
+  print(i)
+  if(i == 5) break
+  i = i + 1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

در برنامه زیر اعداد تصادفی یکنواخت تولید شده تا وقتی کوچکتر از ۰/۵ باشند چاپ می شوند و در صورتیکه مقدار عدد بزرگتر از ۰/۵ باشد حلقه شکسته می شود.

```
>  repeat{
+   g <- runif(1)
+   if (g > 0.5) break
+   cat(g, "\n")
+  }
0.00518448185175
0.28287161886692
0.42854575905948
```

در برنامه زیر تا وقتی i **کوچکتر مساوی ۱۰ باشد محاسبه و چاپ نتایج ادامه می یابد.**

```
> i = 1
> while(i <= 10){
    print(i*i)
    i = i + sqrt(i)
 }
[1] 1
[1] 4
[1] 11.656
```

در دو برنامه زیر جذر عدد ۱۲۳۴۵ به روش نیوتن و بوسیله حلقه های *while* **و** *repeat* **محاسبه می شود.**

$$x = \sqrt{y} \quad \rightarrow \quad y = x^{۲}$$

```
> y <- 12345
> x <- y/2
> repeat{
+ x <- (x + y/x)/2
+ if(abs(x*x-y) < 1e-10) break
+ }
> x
[1] 111.1081

> x <- y/2
> while(abs(x*x-y) > 1e-10) x <- (x + y/x)/2
> x
[1] 111.1081
> x^2
[1] 12345
```

اگر بخواهیم تعداد تکرارهای حلقه *repeat* **تا محاسبه ریشه دوم را تعیین کنیم کافیست شمارنده ای به شکل زیر به برنامه اضافه کنیم.**

```
>  y <- 12345
>  x <- y/2
>  k <- 0
>  repeat{
+  x <- (x + y/x)/2
+  k <- k + 1
+  if(abs(x*x-y) < 1e-10) break
+  }
>  k
[1] 10
```

تابع زیر ریشه p **ام عدد ورودی** y **را بر اساس فرمول بازگشتی نیوتن** $x_{i+۱} = \left((p-۱)x_i + y_i/x_i^{p-۱}\right)/p$ **محاسبه می کند.**

```
> my.root <- function(y, p){
    x <- y/p
    repeat{
      x <- ((p-1)*x + y/x^(p-1))/p
      if(abs(x^p-y) < 1e-10) break
     }
    x
  }
> my.root(y = 10, p = 4)
[1] 1.778279
```

مثال هایی برای آشنایی و تسلط بیشتر در استفاده از دستورات و توابع معرفی شده

تابعی که اعداد زوج و فرد کوچکتر از عدد ورودی را ارائه می نماید.

```
> even.odd <- function(n = 10){
```

```
  x <- seq(from = 0, to = n)
  evens <- x[x %% 2 == 0]
  odds <- x[x %% 2 == 1]
  return(evens,odds)
  }
> even.odd()
```

تابعی که اعداد مضرب ۳ کوچکتر از عدد ورودی را نمایش می دهد.

```
> mazrab3 <- function(n){
  x <- seq(1:n)
  y <- x[x %% 3 == 0]
  return(y)
  }
> mazrab3(45)
 [1]  3  6  9 12 15 18 21 24 27 30 33 36 39 42 45
```

تابعی که اعداد زوج و فرد متغیر ورودی را مشخص و تعداد آنها را چاپ می کند.

```
> zf <- function(x){
  odd <- x[x %% 2 == 1]
  even <- x[x %% 2 == 0]
  odd.n <- length(odd)
  even.n <- length(even)
  list(odd.n = odd.n, even.n = even.n, odd = odd, even = even)
 }
> zf(rpois(20, 5))
```

تابعی که عدد ورودی را برحسب ثانیه دریافت و آنرا بر حسب ساعت، دقیقه و ثانیه نمایش می دهد.

```
> hour <- function(x){
  if(x == 0) return(0)
  a <- abs(round(x))
  h <- floor(a / 3600)
  m <- floor(a %% 3600 / 60)
  s <- a - h * 3600 - m * 60
  list(hour = h, Minutes = m, seconds = s)
 }
```

تابعی که بدون استفاده از تابع *max* مقدار ماکزیمم یک بردار را مشخص می کند.

```
> my.max <- function(x){
   if(length(x) == 1) max <- x
    else {
     max <- x[1]
      for(i in 2:length(x))
       if( x[i] > max ) max <- x[i]
     }
   list(max=max)
  }
```

تابعی که بدون استفاده از تابع *min* مقدار مینیمم یک بردار را مشخص می کند.

```
> my.min_function(x){
  n_length(x)
  if(n == 1) min_x
    else {
     m_x[1]
       for (i in 2:n)
        if(x[i] < m) m_x[i]
     }
   return(m)
  }
```

تابعی برای استفاده از حلقه *for* در محاسبه ماکزیمم متحرک

```
> mov.max = function(x){
    n = length(x)
    y = rep(NA, n)
    for(i in 2:n) y[i] = max(x[i - 1], x[i])
    return(y)
  }
> mov.max(c(3,4,7,2,9,7,8,5,2,8,7,9))
 [1] NA  4  7  7  9  9  8  8  5  8  8  9
```

تابعی برای بکارگیری یک تابع درون تابعی دیگر برای محاسبه $z = \frac{n(\bar{x}+\bar{y})}{s_x+s_y}$ **به صورت زیر است.**

```
> inside <- function(x,y){
    n <- length(x)
       mk <- function(x, y){
         v1 <- mean(x)
         v2 <- mean(y)
         v1 + v2
       }
    s1 <- stdev(x)
    s2 <- stdev(y)
    z <- n * mk(x,y) / (s1 + s2)
    z
   }
```

اگر تابعی مانند *meanxy* **قبلاً تعریف شده باشد در توابع دیگر قابل استفاده خواهد بود.**

```
> meanxy <- function(x, y){
   v1 <- mean(x)
   v2 <- mean(y)
   v1 + v2
 }

> inside <- function(x,y){
    n <- length(x)
    s1 <- stdev(x)
    s2 <- stdev(y)
    z <- n * meanxy(x,y) / (s1 + s2)
    z
   }
```

تابعی برای کاربرد همزمان حلقه *for* **و شرط** *if* **و** *else* **در کد گذاری داده ها**

```
> sgn1 <- function(x){
   for(i in 1:length(x)){
    if(x[i] > 0) x[i] <- 1
    else if(x[i] < 0) x[i] <- -1
    else x[i] <- 0
    }
   x
  }

> sgn2 <- function(x){
   x[x > 0] <- 1
   x[x < 0] <- -1
   x
  }

> sgn3 <- function(x){
   ifelse(x > 0, 1, ifelse(x < 0, -1, 0))
  }
```

تابعی برای کاربرد شرط *if* **و تابع** *Recall* **در محاسبه فاکتوریل عدد ورودی**

```
> fact1 <- function(n){
  if(n <= 1) 1
  else n * Recall(n - 1)
  }
> fact1(5)
[1] 120
```

تابعی که برای عدد ورودی بوسیله حلقه *for* فاکتوریل عدد ورودی را محاسبه می کند.

```
> fact2 <- function(x){
   f <- 1
   if(x < 2) return(1)
    for(i in 2:x){
    f <- f * i
   }
  f
 }
> fact2(8)
[1] 40320
> sapply(0:5,fact2)
[1] 1 1 2 6 24 120
```

تابعی که برای عدد ورودی بوسیله حلقه *while* فاکتوریل عدد ورودی را محاسبه می کند.

```
> fact3 <- function(x){
   f <- 1
   t <- x
   while(t > 1){
    f <- f * t
    t <- t - 1
   }
  return(f)
 }
> sapply(0:5,fact3)
[1] 1 1 2 6 24 120
```

تابعی که برای عدد ورودی بوسیله حلقه *repeat* فاکتوریل عدد ورودی را محاسبه می کند.

```
> fact4 <- function(x){
    f <- 1
    t <- x
  repeat {
    if(t < 2) break
    f <- f * t
    t <- t - 1
  }
  return(f)
 }
> sapply(0:5,fact4)
[1] 1 1 2 6 24 120
```

تابعی که برای عدد ورودی بوسیله تابع *cumprod* فاکتوریل عدد ورودی را محاسبه می کند.

```
> fact5 <- function(x) max(cumprod(1:x))
> sapply(0:5,fact5)
[1] 1 1 2 6 24 120
```

تابعی که برای عدد ورودی بوسیله تابع *gamma* فاکتوریل عدد ورودی را محاسبه می کند.

```
> fact6 <- function(x) gamma(x + 1)
> sapply(0:5,fact6)
 [1] 1 1 2 6 24 120
```

تابعی که برای عدد ورودی بوسیله تابع *gamma* فاکتوریل عدد ورودی را محاسبه می کند.

```
> fact7 <- function(x) prod(1:x)
```

```
> sapply(0:5,fact7)
 [1]  1  1  2  6 24 120
```

نکته: بوسیله تابع *dos.time* **می توان عملکرد توابع مختلف محاسبه فاکتوریل را مقایسه کرد.**

تابعی برای محاسبه ترکیب دو عدد ورودی

```
> comb <- function(n, x){
      if(x == 0 | x == n) n.x <- 1
      else {
            xfact <- prod(1:x)
            nfact <- prod(1:n)
            n.xfact <- prod(1:(n - x))
            n.x <- nfact/(xfact * n.xfact)
      }
   n.x
 }
```

تابعی برای کاربرد حلقه *for* **در فیلتر کردن داده های ورودی**

```
> ar <- function(x, phi){
   n <- length(x)
   for(i in 2:n)
    x[i] <- phi * x[i - 1] + x[i]
   x
  }
> ar(x, 0.3)
 [1] 4.0000000 6.200000 7.860000 9.3580000 5.807400 5.742220
     4.722666 3.416800 2.025040 2.607512 3.782254 5.134676
```

تعریف: دنباله ای از اعداد است که عضو n **ام آن با** F_n **نشان داده می شود و به شکل زیر تعریف می شود.**

$$F_n = F_{n-1} + F_{n-2} \qquad n > 1 \qquad F_0 = 0\ ,\ F_1 = 1$$

توابعی برای تولید n **مقدار اولیه دنباله اعداد فیبوناچی**

```
> fib.loop <- function(n){
   a <- 1
   b <- 0
   print(b)
      while(n > 1){
        print(a)
        tmp <- a
        a <- a + b
        b <- tmp
        n <- n - 1
        }
   }
> fib.loop(10)
[1] 0  1  1  2  3  5  8  13  21  34
```

```
> fibo <- function(n){
    fibvals <- numeric(n)
    fibvals[1] <- 0
    fibvals[2] <- 1
      for (i in 3:n){
        fibvals[i] <- fibvals[i-1] + fibvals[i-2]
      }
    return(fibvals)
  }
> fibo(n = 10)
[1] 0  1  1  2  3  5  8  13  21  34
```

```
> fibon <- function(n) {
    if (n == 1) return(0)
    x <- c(0, 1)
    while (length(x) < n) {
        position <- length(x)
        new <- x[position] + x[position - 1]
        x <- c(x, new)
    }
    return(x)
 }
> fibon(10)
 [1]  0  1  1  2  3  5  8  13  21  34
```

تابعی که اعداد اول کوچکتر از مقدار ورودی را بوسیله حلقه *repeat* محاسبه می کند.

```
> primes <- function(n){
        p <- 2:n
        smallp <- 0
        repeat{
                i <- p[1]
                smallp <- c(smallp, i)
                p <- p[p %% i != 0]
                if(i > sqrt(n)) break
        }
        c(smallp[-1], p)
  }
> primes(20)
 [1] 2  3  5  7  11  13  17  19
```

مثال هایی درباره کاربرد تابع *cat* برای ارائه نتایج

برنامه ای که تعداد خطها را قبل از رخداد اولین شیر محاسبه می کند.

```
> tosscoin <- function(){
  coin <- "tails"          # initialize condition
  count <- -1              # get counting right this way
  while(coin == "tails") {
    coin <- sample(c("heads","tails"),1)
    count <- count + 1
   }
  cat("There were", count, "tails before the first heads\n")
 }
> tosscoin()
There were 2 tails before the first heads
```

تابعی که تعداد تکرار حلقه تا رسیدن مجموع به عدد ۱۰۰ را مشخص می کند.

```
> mycalc <- function(){
    sum <- 0
    n <- 0
      while(sum <= 100){
        sum <- sum + rbinom(1,10,0.5)
        n <- n +1
      }
   cat('It took ')
   cat(n)
   cat(' iterations to finish \n')
  }
> mycalc()
It took 20 iterations to finish
```

تابعی برای کاربرد شرط *if* برای تولید اعداد تصادفی از ۳ توزیع مختلف

```
> my.ran <- function(n, distribution, shape){
    if(distribution == "gamma") rgamma(n, shape) else
    if(distribution == "exp") rexp(n) else
    if(distribution == "norm") rnorm(n) else
   stop("Unknown distribution")
  }
> my.ran(100,"gamma",3)
```

تابعی برای محاسبه ضرایب کجی و کشیدگی برای متغیر ورودی

```
> shape <- function(x){
   skewness <- 0
   kurtosis <- 0
   n <- length(x)
    if(n >=2) {
       m <- mean(x[1:n])
       s <- stdev(x[1:n])
       m3 <- sum((x[1:n]-m)^3)
       m4 <- sum((x[1:n]-m)^4)
       if(s > 0) {
         skewness <- m3/s^3
         kurtosis <- m4/s^4
       }
     c("skewness" = skewness, "kurtosis" = kurtosis)
   }
  }
> shape(x)
  skewness   kurtosis
-0.1806691 17.1733961
```

نمونه گیری

برای گرفتن نمونه تصادفی از یک متغیر یا یک سری اعداد از تابع *sample* **استفاده می شود. شکل کلی این تابع بـه صورت زیر است.**

```
> sample(x, size, replace = F, prob)
```

دستور زیر جایگشت تصادفی از اعداد ۱ تا ۱۰ را ارائه می کند.

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

دستور زیر ۱۰ نمونه تصادفی باجایگذاری از توزیع برنولی با پارامتر ۰/۷ انتخاب می کند.

```
> sample(0:1, size = 10, replace = T, prob = c(0.3,0.7))
[1] 1 1 0 0 1 1 0 1 1 0
```

دستور زیر از بین اعداد ۱ تا ۱۰ تعداد ۲۰ نمونه با احتمالات برابر انتخاب می کند.

```
> sample(10, size = 20, replace = T)
 [1]  5  6  5  3  9  8  1  4  2  6  5  4  2  9  7  7  2  1  5 10
```

دستور زیر ۲۴ نمونه باجایگذاری از بین اعداد ۱ تا ۵ با احتمالات مشخص شده انتخاب می کند.

```
> sample(5, size = 24, prob = c(.3,.4,.1,.1,.1), replace = T)
 [1] 5 2 3 3 4 1 2 2 3 4 1 1 5 3 5 2 2 2 1 2 2 1 5 2
```

برای انتخاب ۱۰ نمونه تصادفی باجایگذاری از دستور زیر استفاده می شود.

```
> x <- rpois(30, 10)
> sample(x, size = 10, replace = T)
 [1] 16  10  10  7  10  5  11  15  14  9
```

برای انتخاب ۲۰ نمونه تصادفی با جایگذاری از تابع چگالی با احتمالات مشخص از دستور زیر استفاده می شود.

```
> sample(c(1:4), size = 20, replace = T, c(0.20,0.15,0.55,0.10))
 [1] 4 3 3 1 1 2 3 1 1 3 3 4 3 3 3 3 3 2 1 1
```

تابعی برای مطالعه میزان پوشش حدود چیبیشف در توزیع تی استودنت

$$P(|X-\mu|<k\sigma)\geq ۱-\frac{۱}{k^۲} \qquad k>۱$$

```
> cheby.t <- function(df, iter, k){
      a <- vector()
      for(i in 1:iter) a[i] <- rt(1, df)
      b <- a[a < mean(a) + k * stdev(a) & a > mean(a) - k * stdev(a)]
      p <- length(b) / iter
      return(p)
  }
> cheby.t(df=5,iter=100,k=1.5)
[1] 0.92
> cheby.t(df=5,iter=100,k=2)
[1] 0.96
> cheby.t(df=5,iter=100,k=3)
[1] 0.98
```

تابعی برای مطالعه میزان پوشش حدود چیبیشف در توزیع فیشر

```
> cheby.f <- function(df1, df2, iter, k){
      a <- vector()
      for(i in 1:iter) a[i] <- rf(1, df1, df2)
      b <- a[a < mean(a) + k * stdev(a) & a > mean(a) - k * stdev(a)]
      p <- length(b) / iter
      return(p)
  }
> cheby.f(df1=5, df2=5, iter=100, k=1.5)
[1] 0.92
> cheby.f(df1=15, df2=15, iter=100, k=1.5)
[1] 0.92
> cheby.f(df1=5, df2=5, iter=100, k=2)
[1] 0.95
> cheby.f(df1=15, df2=15, iter=100, k=2)
[1] 0.95
```

نکته: کمپ و میدل برای یک توزیع تک نمایی رابطه زیر را پیشنهاد کرده اند.

$$P(|X-\mu|<k\sigma)\geq ۱-\frac{۴}{۹k^۲}$$

تابعی که پانصد مشاهده از توزیع های زیر شبیه سازی نموده و نمودارهای بافت نگار، جعبه ای و چگالی را بـرای داده های شبیه سازی شده رسم می کند.

الف) نرمال استاندارد **ب) نرمال با پارامترهای** $\mu=2,\sigma=3$

ج) بتا با پارامترهای $\alpha=2,\beta=3$ **د) گاما با پارامترهای** $\alpha=2,\beta=3$

ه) یکنواخت با پارامترهای $\alpha=2,\beta=3$

```
> simul1 <- function(){
  z <- rnorm(500)
  z23 <- 2 + sqrt(3)*rnorm(500)
  b <- rbeta(500,2,3)
  g <- 3 * rgamma(500,2,1)
  u <- 2 + (3 - 2) * runif(500)
  par(mfrow = c(5,2))
  hist(z); plot(density(z))
  hist(z23); plot(density(z23))
  hist(b); plot(density(b))
  hist(g); plot(density(g))
  hist(u); plot(density(u))
```

```
}
```

تابعی که پانصد مشاهده از توزیع های زیر شبیه سازی نموده و نمودارهای میله ای و چگالی را برای داده هـای شـبیه سازی شده رسم می کند.

الف) دوجمله ای با پارامترهای $n = 20, p = 0.6$ **ب) دوجمله ای منفی با پارامترهای** $n = 20, p = 0.6$

ج) یکنواخت گسسته

$$f(x) = \begin{cases} 0.20 & x = 1 \\ 0.15 & x = 2 \\ 0.55 & x = 3 \\ 0.10 & x = 4 \end{cases}$$

```
> simul2 <- function(){
  b <- rbinom(500,20,0.6)
  nb <- rnbinom(500,20,0.6)
  u <- sample(c(1:4),size=500,replace=T,prob=c(0.2,0.15,0.55,0.1))
  par(mfrow = c(3,2))
  barplot(b); plot(density(b))
  barplot(nb); plot(density(nb))
  barplot(u); plot(density(u))
 }
```

تابعی که توزیع های g,h **را شبیه سازی می کند.**

```
> ghdist <- function(n,g=0,h=0){
  # generate n observations from a g and h dist.
  x <- rnorm(n)
  if (g > 0) ghdist <- (exp(g*x)-1)*exp(h*x^2/2)/g
  if (g == 0) ghdist <- x*exp(h*x^2/2)
  ghdist
 }
```

تابعی برای استفاده از حلقه *repeat* **در محاسبه تعداد دفعاتی که باید نمونه تصادفی از یـک دسـته کـارت ۵۲ تـایی برای رسیدن به هر چهار خال تک (آس) انتخاب کرد.**

```
> draw.tak <- function(){
        draws <- 0
        tak.drawn <- rep(F, 4)
        repeat {
                draw <- sample(1:52, 1, replace = T)
                draws <- draws + 1
                if(draw %% 13 != 1) next
                tak.drawn[draw %/% 13 + 1] <- T
                if(all(tak.drawn)) break
               }
   cat("It took", draws,"draws to draw all four of the taks!\n")
 }
> draw.tak()
It took 96 draws to draw all four of the taks!
```

تمرین: برنامه فوق را به شکلی تغییر دهید که متوسط تعداد دفعات را محاسبه کند.

رسم نمودارهای آماری

برای رسم نمودار بافت نگار از تابع *hist* استفاده می شود. بوسیله شناسه های *nclass* و *breaks* می توان تعداد طبقات و طول طبقات را به دلخواه تنظیم کرد.

```
> hist(x, nclass = "Sturges")
> hist(x, nclass = 5)
> hist(x, breaks = c(-3,-1.5,0,1,2))
```

انتخاب های دیگر برای شناسه *nclass* عبارات *scott, fd* می باشند.

نکته: استورجز (۱۹۲۶) تعداد طبقات را برابر $k = \lceil \log_2 n + 1 \rceil$ ، اسکات (۱۹۷۹) مقدار بهینه $k = \left\lceil \frac{R \sqrt[3]{n}}{3/49\, s} \right\rceil$ و فریدمن دایاکونیس (۱۹۸۱) مقدار استوار $k = \left\lceil \frac{R \sqrt[3]{n}}{2(IQR)} \right\rceil$ را پیشنهاد کرده اند. روش *FD* بر اساس جایگزینی دامنه میان چارکی به جای انحراف معیار پیشنهاد شده است.

برای رسم نمودار جعبه ای از تابع *boxplot* استفاده می شود. بوسیله شناسه *notch* می توان فرم فرو رفته نمودار جعبه ای را رسم کرد.

```
> boxplot(x)
> boxplot(x, notch = TRUE)
```

توابع *barplot* ، *dotchart* ، *pie* و *stem* برای رسم نمودارهای میله ای، نقطه ای، دایره ای و ساقه و برگ استفاده می شوند.

```
> xx <- rpois(10, lambda=5)
> barplot(xx)
> dotchart(xx)
> pie(xx, explode = T, size = 0.95)
> stem(xx)
```

برای رسم نمودار سری زمانی یا پراکنش از تابع *plot* استفاده می شود. اگر به همراه *plot* یک متغیر تعریف شود نتیجه رسم نمودار سری زمانی خواهد بود و اگر دو متغیر تعریف شود نمودار پراکنش رسم خواهد شد. شناسه *type* برای وصل کردن نقاط روی نمودار استفاده می شود و تابع *abline* برای اضافه کردن یک خط ساده یا خط رگرسیون به نمودار می توان استفاده کرد.

```
> x <- rnorm(25)
> plot(x, type = 'l')
> abline(h = 0)
```

برای انتقال محورها به صفر در تابع *plot* از شناسه *axes=F* از برنامه زیر استفاده می شود.

```
> x <- rnorm(100)
> y <- rnorm(100)
> plot(x, y, axes=F)
> axis(side=1, pos=0)
> axis(side=2, pos=0)
```

تابعی برای شبیه سازی توزیع نرمال آمیخته عبارتست از

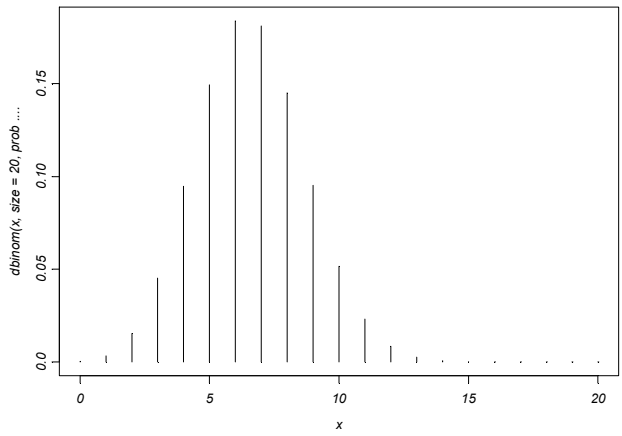
$$f(x) = 0/2\, f_1(x) + 0/8\, f_2(x) \quad , \quad f_1 \sim N(6,4) \ , \ f_2 \sim N(12,4)$$

```
> simul <- function(n = 100, p = 0.2){
   x <- p * rnorm(n,6,2) + (1 - p) * rnorm(n,12,2)
   screen(1); hist(x)
   screen(2); plot(density(x), type="b")
 }
```

نکته: برای تعریف نمودارهای مورد نظر در صفحات مختلف گرافیکی از تابع *screen* استفاده می شود.

رسم نمودار میله ای برای یک توزیع گسسته مانند دوجمله ای

```
> plot(seq(0,20), dbinom(seq(0,20), size=20, prob=0.33), type="h")
```

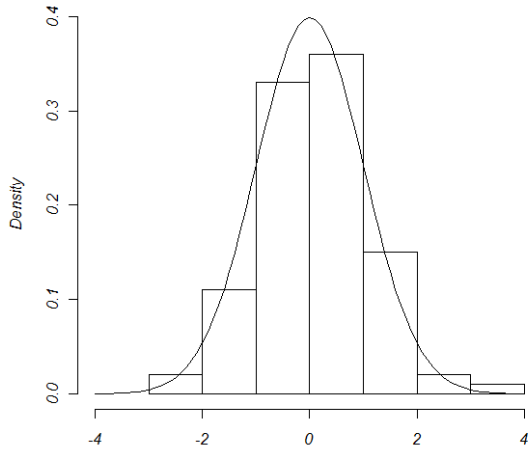


رسم نمودار میله ای برای توزیع های گسسته پواسن و هندسی

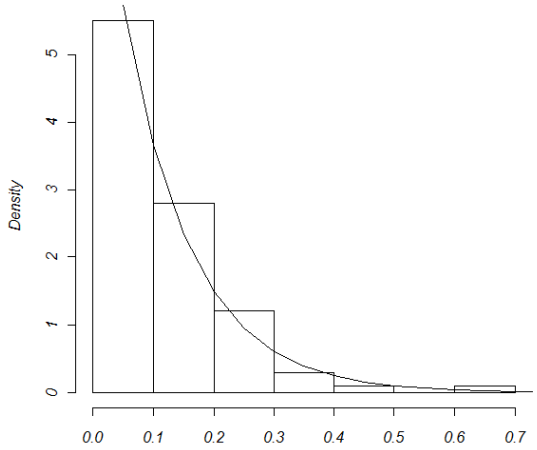
```
> plot(seq(0,20), dpois(seq(0,20), lambda=10), type="h")
> plot(seq(0,20), dgeom(seq(0,20), prob=0.20), type="h")
```

برای رسم نمودار چگالی داده ها بر روی بافت نگار از برنامه زیر استفاده می شود.

```
> x <- rnorm(100)
> hist(x, prob=T, xlim=c(-4,4), ylim=c(0,0.4))
> u <- seq(-4,4,by=0.1)
> lines(u,dnorm(u))
> title('histogram and density')
```

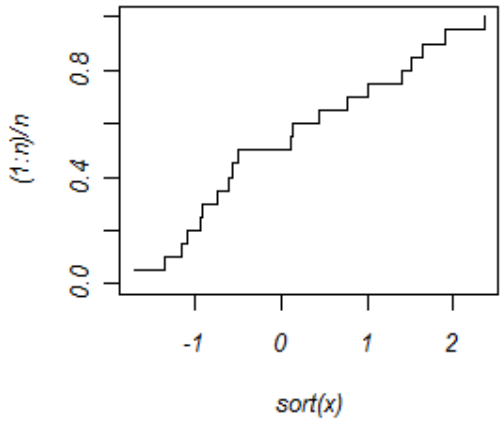



```
> y <- rexp(100,rate=8)
> hist(y, prob=T)
> u <- seq(0,0.9,by=0.05)
> teta <- 1/mean(y)
> lines(u,dexp(u,teta))
> title('histogram and density')
```



برنامه ای برای رسم نمودار تابع توزیع تجربی برای متغیر ورودی

```
> n <- length(x)
> plot(sort(x),(1:n)/n, type="s", ylim=c(0,1))
```



نکته: بـرای تنظیم نمودارهای رسم شـده در صفحات گرافیکی مختلف به ترتیب مورد نظر از تابع *screen* **استفاده می شود. تابع** *tsplot* **برای رسم نمودار سری زمانی استفاده می شود.**

```
> my.plots <- function(x, y){
  par(pty='square')
  screen(1); tsplot(x)
  screen(2); tsplot(x,y)
  }
```

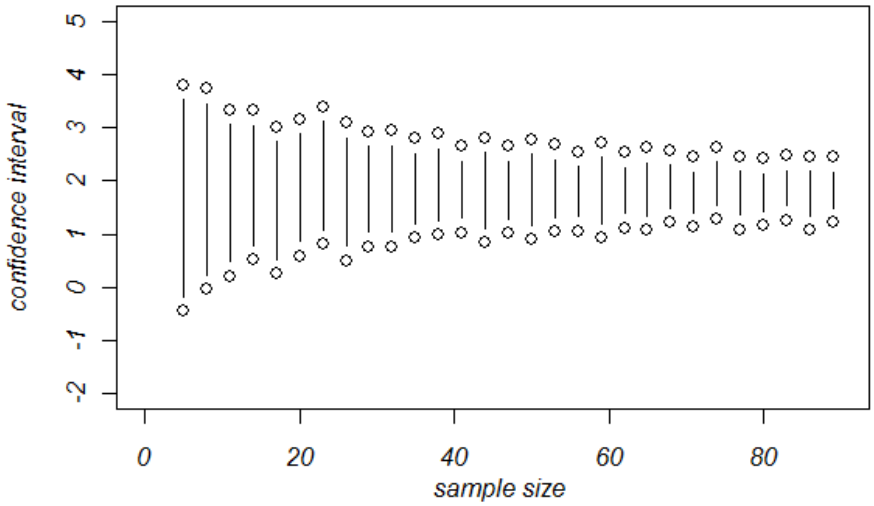
نکته: چارچوب نمودارها مستطیلی است در صورتی که بخواهیم مربعی شود از دستور *par(pty='square')* **می توان استفاده کرد. برای برگرداندن به وضعیت قبل از** *par(pty=' ')* **استفاده می شود. تابع** *par* **برای تقسیم بندی صفحه گرافیکی جهت رسم چند نمودار در یک صفحه نیز بکار می رود. به عنوان مثال برنامه زیر صفحه را به ۴ قسمت تقسیم نموده و نمودارهای بافت نگار، جعبه ای، چگالی و چندک چندک را رسم می کند.**

```
> par(mfrow=c(2,2))
> hist(x)
> boxplot(x)
> iqd <- summary(x)[5] - summary(x)[2]
> plot(density(x, width = 2 * iqd))
> qqnorm(x)
> qqline(x)
```

برای برگرداندن تقسیم بندی صفحه گرافیکی به حالت قبل از دستور زیر استفاده می شود.

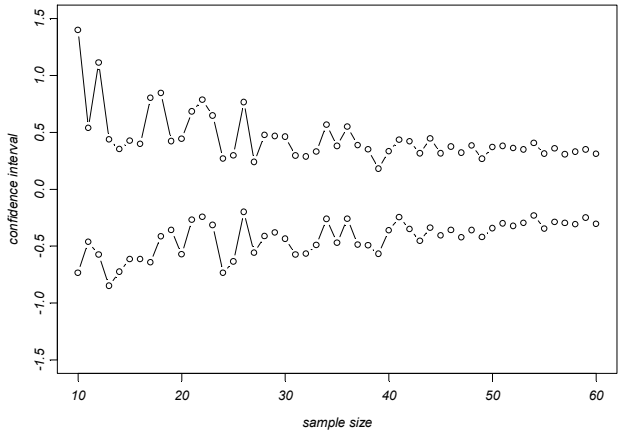
تابعی برای مطالعه تاثیر حجم نمونه روی فاصله اطمینان

```
> conf <- function(nsim = 100){
   x <- rnorm(nsim, 2, 3)
   plot(c(0,100),c(-2,5), type='n', xlab='sample size',
        ylab='confidence interval')
     for(k in seq(5,100,3)){
       a <- numeric(nsim)
        for(i in 1:nsim) a[i] <- mean(sample(x, k, replace = T))
          points(c(k,k), quantile(a, c(0.025,0.975)), type = 'b')
     }
  }
> conf()
```



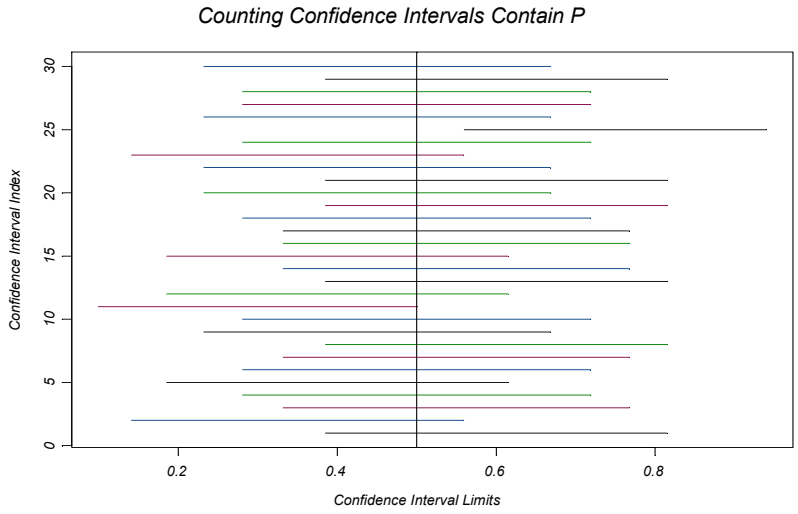
تابعی برای مطالعه رفتار فاصله اطمینان میانگین با افزایش حجم نمونه

```
> confs <- function(n = 60){
    x <- rnorm(n)
    l <- u <- rep(NA, n - 10)
     for(i in 10:n){
       y <- sample(x, size = i)
       l[i] <- mean(y) - 1.96 * stdev(y) / sqrt(i)
       u[i] <- mean(y) + 1.96 * stdev(y) / sqrt(i)
     }
  plot(c(10,n),c(-1.5,1.5), type='n', xlab='sample size',
       ylab='confidence interval')
  lines(l, type='b')
  lines(u, type='b')
 }
> confs()
```



تابع زیر این موضوع که برای یک فاصله اطمینان ۹۵٪ برای پارامتر p انتظار داریم ۹۵ فاصله اطمینان از ۱۰۰ فاصله شامل p باشد، را با رسم نمودار بررسی می کند.

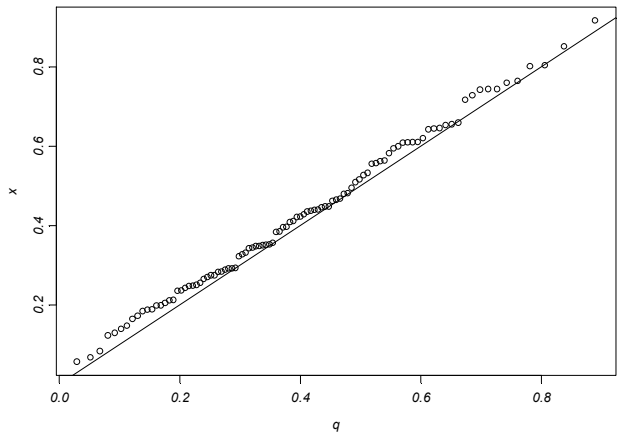
```
> pconfplot <- function(m=30, n=20, p=0.5, alpha=0.05){
  phat <- rbinom(m,n,p)/n
  se <- sqrt(phat*(1-phat)/n)
  z <- qnorm(1-alpha/2)
  lower <- phat-z*se
  upper <- phat+z*se
  matplot(rbind(lower,upper), rbind(1:m,1:m), type='l', lty=1,
  xlab ='Confidence Interval Limits', ylab='Confidence Interval Index',
  main ='Counting Confidence Intervals Contain P')
  abline(v = p)
}
```



تمرین: تابعی برای بررسی دربرگیری ۹۵ فاصله اطمینان از ۱۰۰ فاصله برای μ مشابه مثال قبل بنویسید.

برنامه ای برای تولید ۱۰۰ عداد تصادفی $beta(۲,۳)$ به روش تبدیل متغیر و رسم نمودار چندک چندک بتا

```
> w1 <- rgamma(100, shape = 2, rate = 1)
> w2 <- rgamma(100, shape = 3, rate = 1)
> x <- w1/(w1 + w2)
> q <- qbeta(ppoints(100), 2, 3)
> qqplot(q, x)
> abline(0,1)   # drawing a 45-degree reference line
```



منابع جزوه

شبیه سازی رایانه ای، سیف الله جلیلی

علم و هنر شبیه سازی سیستم‌ها، شانون، رابرت، مرکز نشر دانشگاهی، ۱۳۷۱

شبیه سازی، مولف شلدون راس، مترجمان حسنعلی آذرنوش و حسینعلی نیرومند، انتشارات دانشگاه فردوسی، ۱۳۷۴

شبیه سازی سیستم های گسسته- پیشامد، مولفان جری بنکس و جان باکس، مترجم هاشم محلوجی، موسسه انتشارات علمی، ۱۳۸۴

شبیه سازی، مولف مهندس امین اله مه آبادی، نشر آذرخش، ۱۳۸۶

شبیه سازی کامپیوتری و زبان *SLAM* ، تالیف محمد اقدسی، انتشارات جهاد دانشگاهی صنعتی شریف، ۱۳۷۲

شبیه سازی سیستم‌ها به وسیله کامپیوترهای رقمی، مترجم صالحی فتح آبادی، نشر جهاد دانشگاهی

Arnaud Doucet, Nando de Freitas and Neil Gordon, Sequential Monte Carlo methods in practice, 2001

P. Kevin MacKeown, Stochastic Simulation in Physics, 1997.

Harvey Gould & Jan Tobochnik, An Introduction to Computer Simulation Methods, Part 2, Applications to Physical Systems, 1988.

C.P. Robert and G. Casella. «Monte Carlo Statistical Methods» (second edition). New York: Springer-Verlag, 2004.

R.Y. Rubinstein and D.P. Kroese (2007). «Simulation and the Monte Carlo Method» (second edition). New York: John Wiley & Sons,

N. Metropolis and S. Ulam, «The Monte Carlo Method», Journal of the American Statistical Association, volume 44, number 247, 1949.

Fishman, G.S., (1995) Monte Carlo: Concepts, Algorithms, and Applications, Springer Verlag, New York.

R. E. Caflisch, Monte Carlo and quasi-Monte Carlo methods, Acta Numerica vol. 7, Cambridge University Press, 1998.

Discrete-Event System Simulation, 1986, Jerry Banks, John Scarson

Discrete-Event System Simulation, 2005, 4th edition, Jerry Banks, John Scarson

Simulation Modeling with Pascal, 1959, Robert Okeefe

Introduction to Simulation modeling Using GPSS, 1992

Discrete Event System (Modeling and Performanced Analysis), 1993, Cassandrasm