전산통계학 실습

10. R 프로그래밍

R 수학 함수

- R 에 내장되어 있는 다양한 수학 함수

내장 수학 함수	설명
exp(x)	e의 x승
log(x), log2(x), log10(x)	로그 (밑이 e, 2, 10)
sqrt(x)	제곱근
abs(x)	절댓값
sin(x), cos(x), tan(x)	삼각함수
min(v), max(v)	벡터 내부 최솟값, 최댓값
which.min(v), which.max(v)	벡터 내부 최솟값, 최댓값의 index
pmin(v, ...), pmax(v, ...)	여러 벡터에서 최솟값, 최댓값을 추출한 벡터
sum(v), prod(v)	벡터 내부 원소들의 합과 곱
cumsum(v), cumprod(v)	벡터 내부 원소들의 누적합과 누적곱
round(x), floor(x), ceiling(x)	반올림, 내림, 올림
factorial(x)	팩토리얼

R 수학 함수

- 예제: 확률 계산
  - n 개의 독립적 사건에서 i 번째 사건이 발생할 확률을 p_i 라고 가정한다. 이 사건 중 정확히 하나만 일어날 확률은 얼마일까?

R 수학 함수

- 예제: 확률 계산
  - n 개의 독립적 사건에서 i 번째 사건이 발생할 확률을 p_i 라고 가정한다. 이 사건 중 정확히 하나만 일어날 확률은 얼마일까?
  - (Naive) 만약, n=5 이고 사건의 이름 A, B, C, D, E 라고 한다면, 확률 계산은 다음과 같이 계산될 수 있다.
    - P(사건이 정확히 하나만 발생할 확률) = P(A 발생과 B, C, D, E 발생 X)
      + P(B 발생과 A, C, D, E 발생 X)
      + P(C 발생과 A, B, D, E 발생 X)
      + P(D 발생과 A, B, C, E 발생 X)
      + P(E 발생과 A, B, C, D 발생 X)

R 수학 함수

- 예제: 확률 계산
  - n 개의 독립적 사건에서 i 번째 사건이 발생할 확률을 p_i 라고 가정한다. 이 사건 중 정확히 하나만 일어날 확률은 얼마일까?
  - (Naive) 만약, n=5 이고 사건의 이름 A, B, C, D, E 라고 한다면, 확률 계산은 다음과 같이 계산될 수 있다.
    - P(사건이 정확히 하나만 발생할 확률) = P(A 발생과 B, C, D, E 발생 X)
      + P(B 발생과 A, C, D, E 발생 X)
      + P(C 발생과 A, B, D, E 발생 X)
      + P(D 발생과 A, B, C, E 발생 X)
      + P(E 발생과 A, B, C, D 발생 X)
  - 결론적으로 n 에 대한 확률 계산은 다음과 같은 식으로 구해진다.

$$\sum_{i=1}^{n} p_i(1-p_1)\dots(1-p_n)$$

R 수학 함수

- 예제: 확률 계산
  - n 개의 독립적 사건에서 i 번째 사건이 발생할 확률을 p_i 라고 가정한다. 이 사건 중 정확히 하나만 일어날 확률은 얼마일까?

```
> prob_event_exact_one <- function(p) { # p = prob_vector (size=n)
+   not_p <- 1 - p
+   result <- 0.0
+   for (i in 1:length(p)) {
+     result <- result + p[i] * prod(not_p[-i])
+   }
+   return(result)
+ }
> p <- c(0.3, 0.2, 0.1, 0.25, 0.15)
> prob_event_exact_one(p)
[1] 0.417525
```

  - > not_p
    - 반대인, 사건이 일어나지 않을 확률 $(1 - p_i)$ 를 저장
  - > not_p[-i]
    - 위치 i 의 원소를 제외한 벡터 내부 모든 원소
  - > prod(x)
    - 주어진 벡터 내의 원소들의 곱

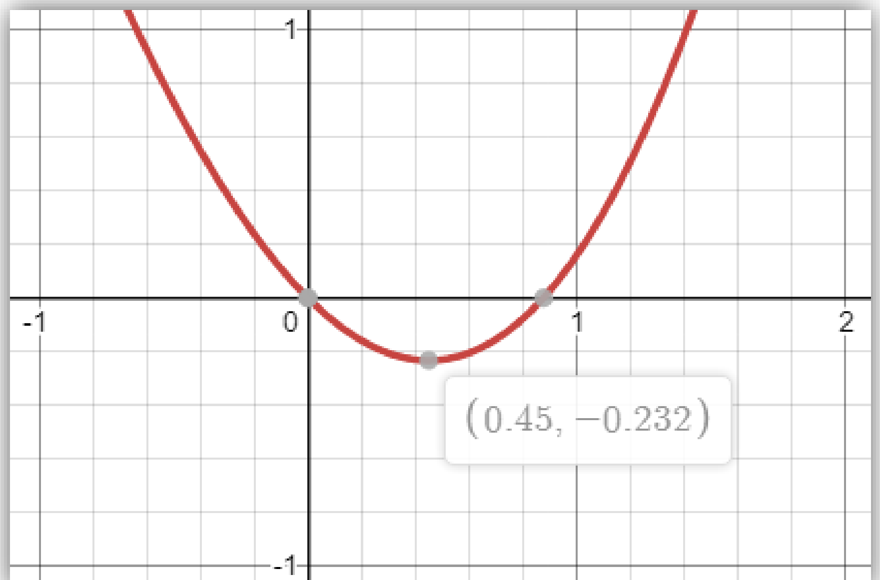
R 수학 함수

- 예제: 최댓값과 최솟값
  - 여러 개의 벡터에서 각 index 단위의 최솟값/최댓값은 무엇인가?
  - > min(x), max(x)
    - 여러 숫자가 저장된 하나의 벡터를 파라미터로 받아 그 중 최솟값/최댓값을 반환
    - 여러 숫자가 저장된 행렬을 하나의 벡터로 보아 값을 찾을 수 있음
  - > pmin(...), pmax(...)
    - 동일 길이를 가진 2개 이상의 벡터에 대하여 index 별 최솟값/최댓값을 반환
    - 동일한 값이 결과로 나오더라도, 하나로 표현
    - 여러 숫자가 저장된 행렬의 행/열별 최솟값/최댓값을 찾을 수 있음

```
> m
     [,1] [,2] [,3]
[1,]   11   14   17
[2,]   12   15   18
[3,]   13   16   19
> min(m)
[1] 11
> min(m[,1], m[,2])
[1] 11
> pmin(m[,1], m[,2])
[1] 11 12 13
```

R 수학 함수

- 예제: 최소화 함수
  - minmize $f(x) = x^2 - \sin x$
  - > nlm(function, default estimate)
    - 주어진 함수에 대한 최솟값 계산
  - 최솟값은 x=0.45 일 때, 약 -0.23 으로 구해짐




```
> nlm(function(x){x^2-sin(x)}, 8)
$`minimum`
[1] -0.2324656

$estimate
[1] 0.4501831

$gradient
[1] 4.024558e-09

$code
[1] 1

$iterations
[1] 5
```

R 수학 함수

- 예제: 미분과 적분
  - > D(expression(x), “x”)
    - “x”에 대하여 주어진 expression(x) 식을 미분 계산
  - > integrate(function(x) {x}, start, end)
    - 주어진 x 에 대한 함수에 대해 start, end 사이로 적분

$$\frac{d}{dx}x^2 = 2x \qquad \int_0^1 x^2 dx \approx 0.333333$$

```
> D(expression(x^2), "x")
2 * x
> D(expression(x), "x")
[1] 1
> D(expression(y^2), "x")
[1] 0
> D(expression(x^2-sin(x)), "x")
2 * x - cos(x)
>
> integrate(function(x){x}, 0, 1)
0.5 with absolute error < 5.6e-15
> integrate(function(x){x^2}, 0, 1)
0.3333333 with absolute error < 3.7e-15
> integrate(function(x){x^2-sin(x)}, 0, 5)
40.95033 with absolute error < 4.6e-13
```

R 수학 함수

- 예제: 선형 대수 연산
  - 벡터와 스칼라의 연산

```
> y <- c(1, 3, 5, 7)
> z <- c(2, 2, 2, 2)
> y
[1] 1 3 5 7
> y*2
[1]  2  6 10 14
> z
[1] 2 2 2 2
> y*z
[1]  2  6 10 14
```

  - 벡터의 내적(inner product, dot product)
    - > crossprod(v1, v2)

```
> x <- c(1, 5, 8)
> y <- c(3, 2, 4)
> x
[1] 1 5 8
> y
[1] 3 2 4
> crossprod(x, y)
     [,1]
[1,]   45
```

R 수학 함수

- 예제: 선형 대수 연산
  - 행렬곱 수행

$$\begin{pmatrix} 3 & 1 \\ 4 & -1 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 3 & -4 \\ 4 & -3 \end{pmatrix}$$

```
> x <- matrix(c(3, 4, 1, -1), nrow=2)
> y <- matrix(c(1, 0, -1, -1), nrow=2)
> x
     [,1] [,2]
[1,]    3    1
[2,]    4   -1
> y
     [,1] [,2]
[1,]    1   -1
[2,]    0   -1
> x %*% y
     [,1] [,2]
[1,]    3   -4
[2,]    4   -3
```

R 수학 함수

- 예제: 선형 대수 연산
  - 선형 방정식 풀이 및 역행렬 찾기

$$\begin{cases} 3x + y = 4 \\ 2x - y = -1 \end{cases}$$

$$\begin{pmatrix} 3 & 1 \\ 2 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 4 \\ -1 \end{pmatrix}$$

```
> m <- matrix(c(3, 2, 1, -1), nrow=2)
> m
     [,1] [,2]
[1,]    3    1
[2,]    2   -1
> a <- c(4, -1)
> solve(m, a)
[1] 0.6 2.2
> solve(m)
     [,1] [,2]
[1,]  0.2  0.2
[2,]  0.4 -0.6
```

R 수학 함수

- 예제: 선형 대수 연산
  - 추가 선형 대수 함수

내장 수학 함수	설명
t()	전치 행렬
qr()	QR 분해
chol()	촐레스키 분해
det()	행렬식
eigen()	아이겐밸류/아이겐벡터
diag()	정사각 행렬 대각값 추출

R 수학 함수

- 예제: 선형 대수 연산
  - 벡터의 외적(cross product)

$$\vec{x} = (x_1, x_2, x_3)$$
$$\vec{y} = (y_1, y_2, y_3)$$

$$\vec{x} \times \vec{y} = (x_2 y_3 - x_3 y_2, x_3 y_1 - x_1 y_3, x_1 y_2 - x_2 y_1)$$

$$\begin{pmatrix} i & j & k \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}$$

R 수학 함수

- 예제: 선형 대수 연산
  - 벡터의 외적(cross product)

$$\vec{x} = (3, 2, -1)$$
$$\vec{y} = (1, -1, 1)$$

$$\vec{x} \times \vec{y} = ?$$

```
> real3dcrossprod <- function(x, y) {
+   m <- rbind(rep(NA,3),x,y)
+   result <- vector(length=3)
+   for (i in 1:3) {
+     result[i] <- -(-1)^i * det(m[2:3, -i])
+   }
+   return(result)
+ }
> x <- c(3, 2, -1)
> y <- c(1, -1, 1)
> x
[1]  3  2 -1
> y
[1]  1 -1  1
> real3dcrossprod(x, y)
[1]  1 -4 -5
```

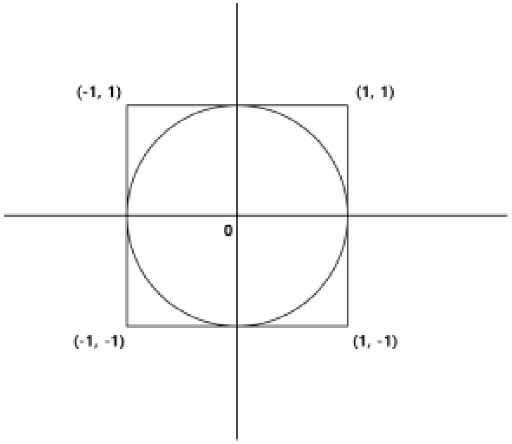
R 시뮬레이션 코딩

- 시뮬레이션(simulation)
  - 어떤 사건과 행동에 대한 결과를 예측(추정)하기 위해 수행
  - 프로그래밍을 통해 조건에 따른 상황들을 현실과 같이 구현
  - 실제 실험하기에 어려운 다양한 상황들을 시뮬레이션으로 구현
    - 자동차 사고 시뮬레이션
    - 네트워크 패킷 생성 시뮬레이션

- 확률 계산 시뮬레이션 (몬테카를로 시뮬레이션)
  - 여러 가지 상황들에 대해 확률적 모델 예측 혹은 실제 상황 등을 가정
  - 조건 적용 및 결과 도출에서 다양한 수학함수 및 계산이 필요

R 시뮬레이션 코딩

- 예제 1.
  - 원주율을 구하는 몬테카를로 시뮬레이션



$$\frac{(\text{원의 넓이})}{(\text{정사각형의 넓이})} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

$$= \frac{(\text{원 내부에 존재하는 점의 수})}{(\text{생성된 전체 점의 수})}$$

- 원주율은 어떤 원이 그 원을 가두는 정사각형과의 비율을 의미한다.
- 그림과 같이 원점을 중심으로 한 변의 길이가 2인 정사각형 평면을 생각한다.
- 위 평면에 존재하는 수많은 임의의 (x, y) 를 생성한다.
- 생성된 임의의 점들에서 원 내부에 존재하는 점의 비율을 구한다.
- 위 비율은 넓이가 4인 정사각형에 대한 원의 면적이므로 넓이 4를 곱한다.
- 더 많은 숫자를 생성할수록 원주율에 근사하게 된다.

R 시뮬레이션 코딩

- 예제 1.
  - 원주율을 구하는 몬테카를로 시뮬레이션
    - > runif(size, min, max)
      - 주어진 값 사이에서 난수(float)를 생성하는 함수
    - 어떤 (x, y) 가 원 내부에 존재한다면, $\sqrt{x^2 + y^2} < 1$ 을 만족합니다.

```
> pi_simulation <- function(n) {
+   x <- runif(n, -1, 1)
+   y <- runif(n, -1, 1)
+   in_circle <- ifelse(sqrt(x^2+y^2) < 1, 1, 0)
+   return(sum(in_circle) / n * 4)
+ }
>
> pi_simulation(100)
[1] 3.56
> pi_simulation(1000)
[1] 3.172
> pi_simulation(10000)
[1] 3.108
> pi_simulation(100000)
[1] 3.14324
```

R 시뮬레이션 코딩

- 예제 2.
  - 공장에서 제품을 생산하고 있다.
  - 아래와 같은 조건을 따르는 공장이 있다.
    - 공장에는 생산 라인 1,000 개가 존재한다.
    - 올바르게 동작하는 생산 라인 하나는 1 시간에 10 개의 제품을 생산한다.
    - 하나의 제품이 만들어질 때, 결함이 생길 확률은 1% 이다.
    - 결함 제품이 하나라도 발견되면, 그 생산 라인은 다음 시간까지 생산을 중단한다.
    - 예를 들어, 11시 XX분에 결함이 발견된 생산 라인은 12시에 다시 생산을 시작한다.
  - 이 공장에서 하루에 생산할 수 있는 평균 제품 개수는?
    - 모두 올바르게 동작하면 24시간 * 10개 * 1,000라인 = 240,000 개 생산 가능

```
> simulation(100)
[1] 227194.2
```

R 시뮬레이션 코딩

- 예제 2.
  - 하나의 제품에 대한 구현
    - 제품 하나 생성이 1% 확률의 결함률을 가질 수 있도록 한다.
    - sample 함수를 이용하면 1:100 에서 2 보다 작은 값으로 실패를 결정할 수 있다.

```
gen_product <- function() {
  return(ifelse(sample(1:100, 1) < 2, "Fail", "Success"))
}
```

R 시뮬레이션 코딩

- 예제 2.
  - 하나의 생산 라인에 대한 구현
    - > replicate(size, function())
      - 주어진 function 을 size 만큼 반복하여 그 결과를 반환하는 함수
      - 구현된 함수로부터 단순한 반복을 필요로 할 때, 기존 반복문을 이용하지 않는 방법

```r
line_fail_update <- function(data) {
  if ("Fail" %in% data) {
    data <- ifelse(min(which(data=="Fail")) > which(data==data), "Success", "Fail")
  }
  return(data)
}

oneday_line <- function() {
  oneday <- 24 * 10
  results <- replicate(oneday, gen_product())
  m_results <- matrix(results, nrow=24, byrow=TRUE)
  updated_results <- t(apply(m_results, 1, line_fail_update))
  count <- sum(ifelse(updated_results == "Success", 1, 0))
  return(count)
}
```

R 시뮬레이션 코딩

- 예제 2.
  - 하나의 생산 라인에 대한 구현
    - > replicate(size, function())
      - 주어진 function 을 size 만큼 반복하여 그 결과를 반환하는 함수
      - 구현된 함수로부터 단순한 반복을 필요로 할 때, 기존 반복문을 이용하지 않는 방법
    - 시뮬레이션이기 때문에, 전체에 대한 상황을 수행하고 후처리가 가능

```
line_fail_update <- function(data) {
  if ("Fail" %in% data) {
    data <- ifelse(min(which(data=="Fail")) > which(data==data), "Success", "Fail")
  }
  return(data)
}

oneday_line <- function() {
  oneday <- 24 * 10
  results <- replicate(oneday, gen_product())
  m_results <- matrix(results, nrow=24, byrow=TRUE)
  updated_results <- t(apply(m_results, 1, line_fail_update))
  count <- sum(ifelse(updated_results == "Success", 1, 0))
  return(count)
}
```

R 시뮬레이션 코딩

- 예제 2.
  - 최종 시뮬레이션 수행
    - 하루 공장 생산을 계산하는 함수를 구현
    - 원하는 수만큼 반복하여 평균을 산출
    - 조건이 존재하지 않는 전체에서 비율을 구하면, 확률로 계산될 수 있음

```r
oneday_factory <- function() {
  results <- replicate(1000, line())
  return(sum(results))
}

simulation <- function(n) {
  return(mean(replicate(n, oneday_factory())))
}

simulation(100)
```

과제

- 주사위 게임에서의 평균 승률
  - Alice와 Bob이 주사위를 던져 내기를 하고자 한다.
    - 두 사람은 각각 5,000원을 가지고 게임을 시작한다.
    - 한 사람이 돈을 모두 잃으면 게임이 종료된다. (다른 사람은 10,000원 이상이 된다.)
  - 둘은 아래와 같이 서로 다른 보상 규칙 두 가지 중 하나를 각각 선택하여 게임을 진행한다.
    - 규칙A: 주사위의 눈이 6의 약수인 경우 상대방에게서 500원을 받는다.
    - 규칙B: 이전 주사위의 눈과 현재 주사위의 눈이 모두 짝수이거나 모두 홀수이면 상대방에게서 1,000원을 받는다. 규칙A와 동시에 발생하는 경우 규칙B가 승리한다. (특히, 규칙B는 첫 번째 게임에서 이전 주사위 눈이 없으므로 승리할 수 없다.)
  - Alice는 위 두 개의 규칙 중 승률이 더 높은 규칙을 먼저 선택하려고 한다. 어떤 규칙을 사용하는 것이 평균 승률이 더 높을지 시뮬레이션을 100,000번 실행하여 각각 계산하고 출력한다.

```
> repgame(100000)
[1] 0.00067
[1] 0.99933
```

과제 제출

- 제출 목록
  - 작성한 코드 파일(.R)
  - 결과 출력 화면 (.PDF)
    - 터미널 캡처(이미지)를 Word 혹은 HWP 에 붙여넣어 PDF 로 변환

- 제출 방법
  - 위 목록의 파일들을 압축
  - 아래 서식으로 압축파일 이름 지정
  - 블랙보드를 통해 제출

- 제출 서식 (XX = 주차번호 ex. 01, 02, ...)
  - 파일 이름: 전산통계학_실습과제_XX주차_학번_이름.zip