R Textbook Companion for Elementary Number Theory by David M. Burton[1]

Created by
Sneha Ranjeet Sonaye
B.E.
Computer Science and Engineering
Atharva College of Engineering
Cross-Checked by
R TBC Team

September 1, 2022

[1]Funded by a grant from the National Mission on Education through ICT - http://spoken-tutorial.org/NMEICT-Intro. This Textbook Companion and R codes written in it can be downloaded from the "Textbook Companion Project" section at the website - https://r.fossee.in.

Book Description

Title: Elementary Number Theory

Author: David M. Burton

Publisher: Mcgraw-hill,1221 Avenue Of The Americas, New York

Edition: 7

Year: 2011

ISBN: 978-0-07-338314-9

R numbering policy used in this document and the relation to the above book.

Exa Example (Solved example)

Eqn Equation (Particular equation of the above book)

For example, Exa 3.51 means solved example 3.51 of this book. Sec 2.3 means an R code whose theory is explained in Section 2.3 of the book.

Contents

List of R Codes

Chapter 1

PRELIMINARIES

R code Exa 1.1 Second Principle of Finite Induction

```
#page 6
a1 <- 1
a2 <- 3
arr <- array(c(a1, a2))
n <- 1
while (n <= 9) {
  if (n >= 3 && n <= 9) {
    arr[n] <- arr[n - 1] + arr[n - 2]
  }
  n <- n + 1
}
n <- n - 1
c <- 0
while (n > 0) {
  if (isTRUE(arr[n] < ((7 / 4) ^ n))) {
    c <- c + 1
  }
  n <- n - 1
}
if (isTRUE(c == 9))
  print("Hence proved")
```

Chapter 2

DIVISIBILITY THEORY IN THE INTEGERS

R code Exa 2.2 The greatest common divisor

```
#page 21
print_divisors <- function(x) {
  if (x < 0) {
    x <- x * (- 1)
  }
  for (i in 1 : x) {
    if ((x %% i) == 0) {
      print(i)
    }
  }
}
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(- x)
```

```
  }else {
    return(x)
  }
}
print_divisors(-12)
print_divisors(30)
print(gcd(-12, 30))
print(gcd(-5, 5))
print(gcd(8, 17))
print(gcd(-8, -36))
```

R code Exa 2.3 the Euclidean Algorithm

```
#page 27
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0)
    return(- x)
  else
    return(x)
}
print(gcd(12378, 3054))
```

R code Exa 2.4 Applying the Euclidean Algorithm to the linear Diophantine equation

```
#page 35
gcd <- function(x, y) {
  while(y) {
```

```
    temp = y
    y = x %% y
    x = temp
  }
  if(x<0)
    return(-x)
  else
    return(x)
}
print(gcd(172,20))
```

Chapter 3

PRIMES AND THEIR DISTRIBUTION

R code Exa 3.1 for determining the canonical form of an integer

```
#page 45
a <- 2093
prime_factors <- vector()


canonical_form <- function(a) {
  y <- ceiling(sqrt(a))
  arr <- prime_numbers(y)
  p <- new_y(a, arr)
  return(p)
}

prime_numbers <- function(n) {
  if (n >= 2) {
    x <- seq(2, n)
    prime_nums <- c()
    for (i in seq(2, n)) {
      if (any(x == i)) {
        prime_nums <- c(prime_nums, i)
```

```
        x <- c(x[(x %% i) != 0], i)
      }
    }
    return(prime_nums)
  }
}

new_y <- function(n, ar) {
  for (i in ar) {
    if (n %% i == 0) {
      break ()
    }
  }
  return(i)
}

check_prime <- function(h) {
  flag <- 0
  if (h > 1) {
    flag <- 1
    for (i in 2 :(h - 1)) {
      if ((h %% i) == 0) {
        flag <- 0
        break
      }
    }
  }
  if (h == 2) {
    flag <- 1
  }
  if (flag == 1) {
    return(TRUE)
  }else {
    return(FALSE)
  }
}

while (isFALSE(check_prime(a))) {
```

```
  p <- canonical_form(a)
  prime_factors <- c(prime_factors, p)
  a <- a / p
}
prime_factors <- c(prime_factors, a)
print(prime_factors)
```

Chapter 4

THE THEORY OF CONGRUENCES

R code Exa 4.1 seful characterization of congruence modulo n in terms of remainders upon division by n

```
#page 65
n <- 7
find_modulo  <- function(a, b) {
  if (a > b) {
    big <- a
  }else {
    big <- b
  }
repeat {
    r1 <- a %% n
    r2 <- b %% n
    n <- n + 2
    if (r1 == r2) {
      n <- n - 2
      break ()
    }
    if (n == big) {
      break ()
```

```
    }
}
  if (r1 == r2) {
    return(n)
   }else {
    return(0)
   }
}
verify_modulo <- function(p, q, r) {
  r1 <- p %% r
  r2 <- q %% r
  if (r1 == r2) {
    return(TRUE)
  }else {
    return(FALSE)
  }
}
print(find_modulo(-56, -11))
print(verify_modulo(-31, 11, 7))
```

R code Exa 4.3 use congruences in carrying out certain types of computations

```
#page 66
 find_rm <- function(f, d) {
   factorial <- 1
  sum <- 0
  for (n in 1 : f) {
    for (i in 1 : n) {
      factorial <- factorial * i
       }
    if (factorial %% d == 0)
    break ()
    sum <- sum + factorial
  factorial <- 1
```

```
  }
  print(sum %% d)
}
(find_rm(100, 12))
```

R code Exa 4.4 to illustrate With suitable precautions cancellation can be allowed

```
#page 67
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0)
    return(-x)
  else
    return(x)
}
check <- function(p, q, r) {
  cmn <- (gcd(p, q))
  p <- p / cmn
  q <- q / cmn
  if (gcd(cmn, r) == cmn)
    r <- r / cmn
  print(c(p, q, r))
}
check(33, 15, 9)
check(-35, 45, 8)
```

R code Exa 4.5 to illustrate binary exponential algorithm

```
#page 71
library(gmp)
library(binaryLogic)
library(base)
calculate_power_mod <- function(x, y, p) {
  val <- as.integer(vector())
  prod <- 1
  b <- as.binary(y)
  for (j in 1 : 6) {
    val <- append(val, powm(5, 2 ^ j, 131))
  }
  count <- 7
  for (v in b) {
    count <- count - 1
    if (v) {
      prod <- prod * val[count]
    }
  }
  print(prod %% p)
}
calculate_power_mod(5, 110, 131)
```

R code Exa 4.6 a well known test for divisibility by 11

```
#page 72
check_num <- function(num, y) {
  digits <- as.integer(vector())
  while (num > 0) {
    digits <- append(digits, num %% 10)
    num <-  as.integer(num / 10)
  }
  digits <- rev(digits)
  if (y == 9) {
    return(sum(num))
  }
```

```
  else if (y == 11) {
    return(sum(num))
  }
}
sum <- function(d) {
  s <- 0
  for (v in d) {
    s <- s + v
  }
  if (s %% 9 == 0) {
    return(TRUE)
  }
  return(FALSE)
}
al_sum <- function(d) {
  s <- 0
  for (v in d) {
    if (v %% 2 == 0) {
      s <- s - v
    }else {
      s <- s + v
    }
  }
  if (s %% 11 == 0) {
    return(TRUE)
  }
  return(FALSE)
}
print(check_num(1571724, 9))
print(check_num(1571724, 11))
```

R code Exa 4.7 solving linear congurrences

```
#page 77

```

```
find_x <- function(a, p, q) {
  x <- as.integer(vector())
  s <- gcd(a, q)
  if (p %% s == 0) {
    i <- q / s
    while (s > 0) {
      t <- (4 + i * s) %% q
      x <- append(x, t)
      s <- s - 1
    }
    x <- sort(x)
    return(x)
  }
}
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(-x)
  }else {
    return(x)
  }
}

print(find_x(18, 30, 42))
```

R code Exa 4.8 solve the linear congruence

```
#page 77
find_x <- function(a, p, q) {
  x <- as.integer(vector())
  s <- gcd(a, q)
```

```
  if (p %% s == 0) {
    a <- a / 3
    p <- p / 3
    q <- q / 3
    a <- a * 7
    p <- p * 7
    a <- 1
    p <- 9
    for (s in 0 : 2) {
      t <- p + q * s
      x <- append(x, t)
    }
    x <- sort(x)
    return(x)
  }
}
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(-x)
  }else {
    return(x)
    }
}

print(find_x(9, 21, 30))
```

R code Exa 4.9 solving linear congruences using Chinese Remainder Theorem

```
#page 80
```

```
find_x <- function(p1, p2, p3, q1, q2, q3) {
  n <- q1 * q2 * q3
  n1 <- n / q1
  n2 <- n / q2
  n3 <- n / q3
  x1 <- find_x0(n1, 1, q1)
  x2 <- find_x0(n2, 1, q2)
  x3 <- find_x0(n3, 1, q3)
  x <- p1 * n1 * x1 + p2 * n2 * x2 + p3 * n3 * x3
  return(x %% n)
}
find_x0 <- function(n, a, q) {
  for (x in 1 : 9) {
    if (((n * x) %% q) == a) {
      return(x)
    }
  }
  }
print(find_x(2, 3, 2, 3, 5, 7))
```

Chapter 5

FERMATS THEOREM

R code Exa 5.1 concrete example of Wilsons theorem

```
#page 94
prove_wilsons_theorem <- function(p) {
  l <- factorial(p - 1)
  if ((l + 1) %% p == 0) {
    return(TRUE)
  }else {
    return(FALSE)
  }
}
print(prove_wilsons_theorem(13))
```

R code Exa 5.2 To illustrate the application of Fermats method

```
#page 98
fermat_factorization <- function(n) {
  n <- as.integer(n)
  lb <- ceiling(sqrt(n))
  ub <- ((n + 1) / 2) - 1
```

```
  lb <- as.integer(lb)
  ub <- as.integer(ub)
  for (k in lb : 352) {
    f <- sqrt(k ^ 2 - n)
    if (perfect(f)) {
      factors <- c(k + f, k - f)
      return(factors)
    }
  }
}
perfect <- function(a) {
  b <- floor(a)
  if ((a / b) == 1) {
    return(TRUE)
  }else {
    return(FALSE)
  }
}
print(fermat_factorization(119143))
```

R code Exa 5.3 factor the positive integer using the Euclidean Algorithm

```
#page 100
factorize <-  function(n) {
  uy <- floor(sqrt(n))
  ux <- floor(n / 2)
  for (xn in ux : uy) {
    for (yn in uy : 1) {
      c <- xn ^ 2 - yn ^ 2
      m <- c %% n
      if (m == 0) {
        ans <- c(gcd(xn - yn, n), gcd(xn + yn, n))
        return(ans)
      }
    }
```

```
  }
}
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(-x)
  }else {
    return(x)
  }
}
print(factorize(2189))
```

R code Exa 5.4 factorization method by Maurice Kraitchik

```
#page 100
factorize <- function(n) {
  uy <- floor(sqrt(n))
  ux <- floor(n / 2)
  for (xn in ux : uy) {
    for (yn in uy : 1) {
      c <- xn ^ 2 - yn ^ 2
      m <- c %% n
      if (m == 0) {
        ans <- c(gcd(xn - yn, n), gcd(xn + yn, n))
        return(ans)
      }
    }
  }
}
gcd <- function(x, y) {
  while (y) {
```

```
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(- x)
  } else {
    return(x)
  }
}
print(factorize(12499))
```

Chapter 6

NUMBER THEORETIC FUNCTIONS

R code Exa 6.1 to find the sum of positive divisors of n

```
#page106
library(collections)
solve <- function(n) {
  p <- vector()
  k <- vector()
  i <- 0
  while (n %% 2 == 0) {
    i <- i + 1
    n <- n / 2
  }
  if (i != 0) {
    p <- append(p, 2)
    k <- append(k, i)
  }
  for (num in 3 : sqrt(n)) {
    if (num %% 2 == 1) {
      i <- 0
      while (n %% num == 0) {
        i <- i + 1
```

```
        n <- n / num
      }
      if (i != 0) {
        p <- append(p, num)
        k <- append(k, i)
      }
    }
  }
  tau <- no_of_divisors(k)
  print(tau)
  sigma <- sum_of_divisors(p, k)
  print(sigma)
}
sum_of_divisors <- function(p, k) {
  sum <- 1
  c <- length(p)
  for (x in 1 : c) {
    sum <- sum * (((p[x] ^ (k[x] + 1)) - 1) / (p[x]
       - 1))
  }
  return(sum)
}
no_of_divisors <- function(k) {
  no <- 1
  for (x in k) {
    no <- no * (x + 1)
  }
  return(no)
}
solve(180)
```

R code Exa 6.2 to find the number of zeros with which the decimal representation of 50 factorial terminates

```
#page 118
```

```
n <- 50
pow_of_2 <- 0
pow_of_5 <- 0
for (v in 1 : 5) {
  pow_of_2 <- pow_of_2 + floor(n / (2 ^ v))
}
print(pow_of_2)
for (v in 1 : 2) {
  pow_of_5 <- pow_of_5 + floor(n / (5 ^ v))
}
print(pow_of_5)
```

R code Exa 6.3 to clarify a Corollary

```
#page 120
n <- 6
tau <- 0
sigma <- 0
for (num in 1 : n) {
  tau <- tau + floor(n / num)
}
print(tau)
for (num in 1 : n) {
  sigma <- sigma + (num * floor(n / num))
}
print(sigma)
```

R code Exa 6.4 to calculate the day of the week on which March 1 1990 fell

```
#page 125
c <- 19
y <- 90
```

```
d <- (3 - 2 * c + y + floor(c / 4) + floor(y / 4))
    %% 7
if (d == 0) {
  print("Sunday")
} else if (d == 1) {
  print("Monday")
} else if (d == 2) {
  print("Tuesday")
} else if (d == 3) {
  print("Wednesday")
} else if (d == 4) {
  print("Thursday")
} else if (d == 5) {
  print("Friday")
} else {
  print("Saturday")
}
```

R code Exa 6.5 to calculate on what day of the week will January 14 2020 occur

```
#page 126
m <- 11
d <- 14
c <- 20
y <- 19
w <- (d + floor((2.6) * m - 0.2) - 2 * c + y + floor
    (c / 4) + floor(y / 4)) %% 7
if (w == 0) {
  print("Sunday")
} else if (w == 1) {
  print("Monday")
} else if (w == 2) {
  print("Tuesday")
} else if (w == 3) {
```

```
  print("Wednesday")
} else if (w == 4) {
  print("Thursday")
} else if (w == 5) {
  print("Friday")
} else {
  print("Saturday")
}
```

Chapter 7

EULERS GENERALIZATION OF FERMATS THEOREM

R code Exa 7.1 to calculate phi of a number

```
#page 134
n <- 360
number <- n
p <- vector()
k <- vector()
i <- 0
while (n %% 2 == 0) {
  i <- i + 1
  n <- n / 2
}
if (i != 0) {
  p <- append(p, 2)
  k <- append(k, i)
}
for (num in 3 : sqrt(n)) {
  if (num %% 2 == 1) {
    i <- 0
    while (n %% num == 0) {
      i <- i + 1
```

```
      n  <- n / num
    }
    if (i != 0) {
      p <- append(p, num)
      k <- append(k, i)
    }
  }
}
pos_prime <- function(p, n) {
  sum <- number
  c <- length(p)
  for (x in 1 : c) {
    sum <- sum * (1 - (1 / p[x]))
  }
  return(sum)
  }
phi <- pos_prime(p, n)
print(phi)
```

R code Exa 7.2 to reduce large powers modulo n using Eulers theorem

```
#page 138
calculate <- function(a, r, n) {
  c <- 0
  while (r %% 2 == 0 & r != 0) {
    c <- c + 1
    r <- r / 2
  }
  ans <- a
  for (var in 1 : c) {
    ans <- (ans ^ 2) %% n
  }
   return(ans)
}

```

```
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(- x)
  }else {
    return(x)
  }
}
a <- 3
r <- 256
n <- 100
print(gcd(a, n))
number <- n
p <- vector()
k <- vector()
i <- 0
while (n %% 2 == 0) {
  i <- i + 1
  n <- n / 2
}
if (i != 0) {
  p <- append(p, 2)
  k <- append(k, i)
}
for (num in 3 : sqrt(n)) {
  if (num %% 2 == 1) {
    i <- 0
    while (n %% num == 0) {
      i <- i + 1
      n <- n / num
    }
    if (i != 0) {
      p <- append(p, num)
      k <- append(k, i)
```

```
    }
  }
}
pos_prime <- function(p, n) {
  sum <- number
  c <- length(p)
  for (x in 1 : c) {
    sum <- sum * (1 - (1 / p[x]))
  }
  return(sum)
}
phi <- pos_prime(p, n)
print(phi)
q <- floor(r / phi)
rd <- r %% phi
r <- rd
ans <- calculate(a, r, number)
print(ans)
```

R code Exa 7.3 a numerical example of Gauss theorem

```
#page 142
phi <- function(n) {
  c <- 0
  for (v in 1 : n) {
    if (gcd(v, n) == 1) {
      c <- c + 1
    }
  }
  return(c)
}

gcd <- function(x, y) {
  while (y) {
    temp <- y
```

```
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(- x)
  }else {
    return(x)
  }
}
n <- 10
d <- vector()
for (m in 1 : n) {
  d <- append(d, gcd(m, n))
}
d <- unique(d)
sum_phi <- 0
for (v in d) {
  sum_phi <- sum_phi + phi(v)
}
print(sum_phi == n)
```

R code Exa 7.4 an example of theorem 7 7

```
#page 143
n <- 30
number <- n
p <- vector()
k <- vector()
i <- 0
while (n %% 2 == 0) {
  i <- i + 1
  n <- n / 2
}
if (i != 0) {
  p <- append(p, 2)
```

```r
  k <- append(k, i)
}
s <- sqrt(number)
for (num in 3 : s) {
 if (num %% 2 == 1) {
    i <- 0
    while (n %% num == 0) {
      i <- i + 1
      n <- n / num
    }
    if (i != 0) {
      p <- append(p, num)
      k <- append(k, i)
    }
  }
}
pos_prime <- function(p, n) {
  sum <- n
  c <- length(p)
  for (x in 1 : c) {
    sum <- sum * (1 - (1 / p[x]))
  }
  return(sum)
}
phi <- pos_prime(p, number)
rel_prime <- vector()
for (v in 1 : number) {
  if (gcd(v, number) == 1) {
    rel_prime <- append(rel_prime, v)
  }
}
sum <- 0
for (v in rel_prime) {
  sum <- sum + v
}
desired_sum <- (1 / 2) * number * phi
print(isTRUE(all.equal(sum, desired_sum)))
```

Chapter 8

PRIMITIVE ROOTS AND INDICES

R code Exa 8.1 to find the integers that also have order 12 modulo 13

```
#page 149
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(-x)
  }else {
    return(x)
  }
}
n <- 13
ans <- vector()
for (num in 1 : n) {
  for (v in 1 : n) {
    if (((num ^ v) %% n) == 1) {
      ans <- append(ans, v)
```

```
      break ()
    }
  }
}
print(ans)
for (x in 2 : 3) {
  if (ans[2 ^ x] == (ans[2] / gcd(x, ans[2]))) {
     print(TRUE)
    }
}
for (x in 1 : 12) {
  if (gcd(x, 12) == 1) {
    print(x)
  }
}
```

R code Exa 8.3 primitive roots for prime

```
#page 157
primitive_root <- function(g, n) {
  number <- n
  i <- 0
  ptt <- vector()
  while ((n %% 2) == 0) {
    i <- i + 1
    n <- n / 2
  }
  if (i != 0) {
    ptt <- append(ptt, number / 2)
  }
  for (var in 3 : sqrt(number)) {
    if (var %% 2 == 1) {
      i <- 0
      while (n %% var == 0) {
        i <- i + 1
```

```
        n <- n / var
      }
      if (i != 0) {
        ptt <- append(ptt, number / var)
      }
    }
  }
  ptt <- sort(ptt)
for (num in 2 : number) {
  i <- 0
  for (x in ptt) {
    if ((num ^ x) %% g == 1) {
      break ()
    }else {
      i <- i + 1
    }
  }
  if (i == length(ptt)) {
    return(num)
  }
}
}
phi <- function(n) {
number <- n
p <- vector()
k <- vector()
i <- 0
while ((n %% 2) == 0) {
  i <- i + 1
  n <- n / 2
}
if (i != 0) {
  p <- append(p, 2)
  k <- append(k, i)
}
for (num in 3 : sqrt(number)) {
  if (num %% 2 == 1) {
    i <- 0
```

```
    while (n %% num == 0) {
      i <- i + 1
      n <- n / num
    }
    if (i != 0) {
      p <- append(p, num)
      k <- append(k, i)
    }
  }
}
pos_prime <- function(p, n) {
  sum <- number
  c <- length(p)
  for (x in 1 : c) {
    sum <- sum * (1 - (1 / p[x]))
  }
  return(sum)
}
if (length(p) == 0) {
  phi <- number - 1
}else {
  phi <- pos_prime(p, n)
  }
return(phi)
}
ord <- 6
mod <- 31
npr <- phi(ord)
p <- (phi(mod))
pr <- primitive_root(mod, p)
kn <- vector()
for (k in 1 : p) {
  if ((p / gcd(k, p)) == ord) {
   kn <- append(kn, k)
  }
}
for (p in kn) {
  print((pr ^ p) %% mod)
```

```
}
```

R code Exa 8.4 solve congruences using theory of indices

```
#page 165
mod <- function(a, z, l) {
  ans <- vector()
  for (k in 1 : l) {
    if (k %% z == a) {
      ans <- append(ans, k)
    }
  }
  return(ans)
}
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(- x)
  } else {
    return(x)
  }
}
primitive_root <- function(g, n) {
  i <- 0
  number <- n
  ptt <- vector()
  while ((n %% 2) == 0) {
    i <- i + 1
    n <- n / 2
  }
  if (i != 0) {
```

```r
      ptt <- append(ptt, number / 2)
    }
    for (var in 3 : sqrt(number)) {
      if (var %% 2 == 1) {
        i <- 0
        while (n %% var == 0) {
          i <- i + 1
          n <- n / var
        }
        if (i != 0) {
          ptt <- append(ptt, number / var)
        }
      }
    }
    ptt <- sort(ptt)
    for (num in 2 : number) {
      i <- 0
      for (x in ptt) {
        if ((num ^ x) %% g == 1) {
          break ()
        }else {
          i <- i + 1
        }
      }
      if (i == length(ptt)) {
        return(num)
      }
    }
}
phi <- function(n) {
    number <- n
    p <- vector()
    k <- vector()
    i <- 0
    while ((n %% 2) == 0) {
      i <- i + 1
      n <- n / 2
    }
```

```
  if (i != 0) {
    p <- append(p, 2)
    k <- append(k, i)
  }
  for (num in 3 : sqrt(number)) {
    if (num %% 2 == 1) {
      i <- 0
      while (n %% num == 0) {
        i <- i + 1
        n <- n / num
      }
      if (i != 0) {
        p <- append(p, num)
        k <- append(k, i)
      }
    }
  }
  pos_prime <- function(p, n) {
    sum <- number
    c <- length(p)
    for (x in 1 : c) {
      sum <- sum * (1 - (1 / p[x]))
    }
    return(sum)
  }
  if (length(p) == 0) {
    phi <- number - 1
  } else {
    phi <- pos_prime(p, n)
  }
  return(phi)
}
r <- 4
ind_a <- 9
n <- 13
ind <- vector()
a <- vector()
ans_x <- vector()
```

```
phi <- phi(n)
pr <- primitive_root(13, phi)
for (an in 1 : phi) {
  if (gcd(an, n) == 1) {
    for (k in 1 : n) {
      if (((pr ^ k) %% n) == an) {
        ind <- append(ind, k)
        a <- append(a, an)
        break ()
      }
    }
  }
}
indxx9 <- ind[7] - ind[4]
indx <- mod(1, 4, phi)
for (x in a) {
  if (is.element(ind[x], indx)) {
    ans_x <- append(ans_x, x)
  }
}
print(ans_x)
```

R code Exa 8.5 solve congruences

```
#page 166
mod <- function(a, z, l) {
  ans <- vector()
  for (k in 1 : l) {
    if (k %% z == a) {
      ans <- append(ans, k)
    }
  }
  return(ans)
}
gcd <- function(x, y) {
```

```
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(- x)
  } else {
    return(x)
  }
}
phi <- function(n) {
  number <- n
  p <- vector()
  k <- vector()
  i <- 0
  while ((n %% 2) == 0) {
    i <- i + 1
    n <- n / 2
    }
  if (i != 0) {
    p <- append(p, 2)
    k <- append(k, i)
  }
  for (num in 3 : sqrt(number)) {
    if (num %% 2 == 1) {
      i <- 0
      while (n %% num == 0) {
        i <- i + 1
        n <- n / num
      }
      if (i != 0) {
        p <- append(p, num)
        k <- append(k, i)
      }
    }
  }
  pos_prime <- function(p, n) {
```

```
    sum <- number
    c <- length(p)
    for (x in 1 : c) {
      sum <- sum * (1 - (1 / p[x]))
    }
    return(sum)
  }
  if (length(p) == 0) {
    phi <- number - 1
  } else {
    phi <- pos_prime(p, n)
  }
  return(phi)
}
solution <- function(n, a, k) {
  if (gcd(a, n) != 1) {
    print("gcd is not 1")
  }
  phi <- phi(n)
  d <- gcd(k, phi)
  if ((a ^ (phi / d) %% n) == 1) {
    print(paste(d, "Solutions exist"))
  } else {
    print("No solution exists")
  }
}
n <- 13
a <- 4
k <- 3
p <- phi(n)
solution(n, a, k)
a <- 5
solution(n, a, k)
ax <- vector()
ind <- vector()
ans_x <- vector()
for (an in 1 : p) {
  if (gcd(an, n) == 1) {
```

```
    for (c in 1 : n) {
      if (((pr ^ c) %% n) == an) {
        ind <- append(ind, c)
        ax <- append(ax, an)
        break ()
      }
    }
  }
}

a <- 9
n <- 12
a <- (a / k)
n <- n / k
indx <- mod(a, n, p)
for (x in ax) {
  if (is.element(ind[x], indx)) {
    ans_x <- append(ans_x, x)
  }
}
print(ans_x)
```

Chapter 9

THE QUADRATIC RECIPROCITY LAW

R code Exa 9.1 to find quadratic residues and non residues

```
#page 171
n <- 13
residues <- vector()
non_residues <- vector()
for (v in 1 : (n - 1)) {
  residues <- append(residues, (v ^ 2) %% n)
}
residues <- sort(unique(residues))
print(residues)
for (v in 1 : (n - 1)) {
  if (!is.element(v, residues)) {
    non_residues <- append(non_residues, v)
  }
}
print(non_residues)
n_consecutive_pairs <- (1 / 4) * (n - 4 - (- 1) ^ ((
   n - 1) / 2))
print(n_consecutive_pairs)
```

R code Exa 9.2 check residues of a number

```r
#page 172
check_residue <- function(a, p) {
 f <- (a ^ ((p - 1) / 2)) %% p
 if (f == 1 | f == (p - 1)) {
   print(paste(a, "is residue of", p))
 }
}
p <- 13
a <- 2
check_residue(a, p)
a <- 3
check_residue(a, p)
```

R code Exa 9.3 Using the Legendre symbol to display results

```r
#page 176
n <- 13
ls <- vector()
residues <- vector()
non_residues <- vector()
for (v in 1 : (n - 1)) {
  residues <- append(residues, (v ^ 2) %% n)
}
residues <- sort(unique(residues))
for (v in 1 : (n - 1)) {
  if (!is.element(v, residues)) {
    non_residues <- append(non_residues, v)
  }
}
for (var in 1 : (n - 1)) {
```

```
  if (is.element(var, residues)) {
    ls <- append(ls, 1)
  } else {
    ls <- append(ls, - 1)
  }
}
l <- length(ls)
for (var in 1 : l) {
  ans <- sprintf("(%d/%d) = %d", var, n, ls[var])
  print(ans)
}
```

R code Exa 9.4 to check if a congruence is solvable

```
#page 177
find <- function(l, s) {
  if (l < 0) {
    l <- l * (- 1)
  }
  m <- l %% s
  l <- m
  squares <- vector()
  pk <- vector()
  k <- vector()
  i <- 0
  n <- l
  while (n %% 2 == 0) {
    i <- i + 1
    n <- n / 2
  }
  if (i != 0) {
    pk <- append(pk, 2)
    k <- append(k, i)
  }
  for (num in 3 : sqrt(n)) {
```

```
    if (num %% 2 == 1) {
      i <- 0
      while (n %% num == 0) {
        i <- i + 1
        n <- n / num
      }
      if (i != 0) {
        pk <- append(pk, num)
        k <- append(k, i)
      }
    }
  }
  for (x in seq_len(length(k))) {
    if (k[x] == 2) {
      squares <- append(squares, pk[x])
    }
  }
  for (sq in squares) {
    l <- l / (sq ^ 2)
  }
  mod <- ((l ^ ((s - 1) / 2)) %% s)
  if (mod == (s - 1)) {
    return(- 1)
  } else {
    return(mod)
  }
}

a <- -46
p <- 17
l <- -46
s <- 17
ls <- find(l, s)
if (ls == (- 1)) {
  print("No solution")
} else {
  print("solution exists")
}
```

R code Exa 9.5 to prove a Legendre corollary

```
#page 188
solve <- function(p, q) {
  if (p == 2) {
    if (q %% 8 == 1 | q %% 8 == 7) {
      return(1)
    }else if (q %% 8 == 3 | q %% 8 == 5) {
      return(- 1)
    }
  } else {
    t <- p
    p <- q
    q <- t
    p <- p %% q
    solve(p, q)
  }
}
p <- 29
q <- 53
i <- 0
final <- 1
answer <- vector()
squares <- vector()
factors <- vector()
pk <- vector()
px <- vector()
k <- vector()
m1 <- p %% 4
m2 <- q %% 4
if (m1 == m2) {
  if (m1 == 1) {
    t <- p
    p <- q
```

```
    q <- t
  } else {
    t <- p
    p <- q * - 1
    q <- t
  }
}
p <- p %% q
n <- p
while (n %% 2 == 0) {
  i <- i + 1
  n <- n / 2
}
if (i != 0) {
  pk <- append(pk, 2)
  k <- append(k, i)
}
for (num in 3 : sqrt(n)) {
  if (num %% 2 == 1) {
    i <- 0
    while (n %% num == 0) {
      i <- i + 1
      n <- n / num
    }
    if (i != 0) {
      pk <- append(pk, num)
      k <- append(k, i)
    }
  }
}
for (x in seq_len(length(k))) {
  if ((k[x] >= 2) & (k[x] %% 2 == 0)) {
    squares <- append(squares, pk[x])
    px <- append(px, k[x])
  }else if (k[x] == 1) {
    factors <- append(factors, pk[x])
    p <- p / pk[x]
  } else {
```

```
    squares <- append(squares, pk[x])
    px <- append(px, (k[x] - 1))
  }
}
for (sq in squares) {
  for (pw in px) {
  p <- p / (sq ^ pw)
  }
factors <- append(factors, p)
for (f in factors) {
  ans <- solve(f, q)
  answer <- append(answer, ans)
}
for (a in answer) {
  final <- final * a
}
}
print(final)
```

R code Exa 9.6 to find the solution of a quadratic congruence with a composite

```
#page 189
p <- 196
q <- 1357
i <- 0
pk <- vector()
k <- vector()
squares <- vector()
for (q1 in 2 : sqrt(q)) {
  if (q %% q1 == 0) {
    q2 <- q / q1
    break ()
  }
}
```

```
p1 <- p %% q1
n <- p1
while (n %% 2 == 0) {
  i <- i + 1
  n <- n / 2
}
if (i != 0) {
  pk <- append(pk, 2)
  k <- append(k, i)
}
for (num in 3 : sqrt(p1)) {
  if (num %% 2 == 1) {
    i <- 0
    while (n %% num == 0) {
      i <- i + 1
      n <- n / num
    }
    if (i != 0) {
      pk <- append(pk, num)
      k <- append(k, i)
    }
  }
}
for (x in seq_len(k)) {
  if (k[x] == 2) {
    squares <- append(squares, pk[x])
  }
}
for (sq in squares) {
    p1 <- p1 / (sq ^ 2)
}
if (p1 == 3) {
  if (q1 %% 12 == 1 | q1 %% 12 == (12 - 1)) {
  ls1 <- 1
  } else if (q1 %% 12 == 5 | q1 %% 12 == (12 - 5)) {
    ls1 <- -1
  }
}
```

```
p2 <- p %% q2
if (q2 > p2) {
  m1 <- p2 %% 4
  m2 <- q2 %% 4
  if (m1 == m2) {
    if (m1 == 1) {
      t <- p2
      p2 <- q2
      q2 <- t
    }else if (m1 == 3) {
      t <- p2
      p2 <- q2
      q2 <- t
      s <- -1
    }
  }
}
if (p2 > q2) {
  p2 <- p2 %% q2
}
if (p2 == 2) {
  if (q2 %% 8 == 1 | q2 %% 8 == 7) {
    ls2 <- s * 1
  } else if (q2 %% 8 == 3 | q2 %% 8 == 5) {
  ls2 <- s * - 1
}
}
if (ls1 == 1 & ls2 == 1) {
  print(”solvable”)
}
```

Chapter 10

INTRODUCTION TO CRYPTOGRAPHY

R code Exa 10.1 Example of Vigeneres method of crypography using autokey

```
#page 200
library(gtools)
library(readr)
pv <- vector()
kv <- vector()
cv <- vector()
c <- ""
plain_text <- "ONE IF BY DAWN"
p <- gsub(" ", "", plain_text, fixed = TRUE)
seed <- "K"
k <- paste(seed, substr(p, 1, nchar(p) - 1), sep = "")
p_split <- strsplit(p, "")
k_split <- strsplit(k, "")
for (ch in p_split) {
  pv <- append(pv, asc(ch) - 65)
}
for (ch in k_split) {
```

```
  kv <- append(kv, asc(ch) - 65)
}
for (num in seq_len(length(pv))) {
  for (n in seq_len(length(kv))) {
    if (n == num) {
      cv <- append(cv, (kv[n] + pv[num]) %% 26)
    }
  }
}
for (n in cv) {
  num <- n + 65
  c <- paste(c, chr(num), sep = "")
}
c <- sub("\\s+$", "", gsub("(.{3})(.{2})(.{2})", "\\1 \\2 \\3 ", c))
print(c)
```

R code Exa 10.2 To illustrate Hills cipher

```
#page 201
library(gtools)
hill_cipher <- function(block) {
  p1 <- substr(block, 1, 1)
  p2 <- substr(block, 2, 2)
  p1 <- asc(p1) - 65
  p2 <- asc(p2) - 65
  c1 <- (a * p1 + b * p2) %% 26
  c2 <- (c * p1 + d * p2) %% 26
  c <- paste0(chr(c1 + 65), chr(c2 + 65))
  return(c)
}
decrypt <- function(block) {
  c1 <- substr(block, 1, 1)
  c2 <- substr(block, 2, 2)
  c1 <- asc(c1) - 65
```

```
  c2 <- asc(c2) - 65
  p1 <- (da * c1 + db * c2) %% 26
  p2 <- (dc * c1 + dd * c2) %% 26
  p <- paste0(chr(p1 + 65), chr(p2 + 65))
  return(p)
}
a <- 2
b <- 3
c <- 5
d <- 8
bl_v <- vector()
message <- "BUY NOW"
m <- gsub(" ", "", message, fixed = TRUE)
blocks <- sub("\\s+$", "", gsub("(.{2})", "\\1 ", m)
   )
block1 <- substr(blocks, 1, 2)
c1 <- hill_cipher(block1)
block2 <- substr(blocks, 4, 5)
c2 <- hill_cipher(block2)
block3 <- substr(blocks, 7, 8)
c3 <- hill_cipher(block3)
cipher <- paste0(c1, c2, c3)
cipher <- sub("\\s+$", "", gsub("(.{3})", "\\1 ",
   cipher))
print(cipher)
da <- d
db <- -1 * b
dc <- -1 * c
dd <- a
bl_v <- vector()
cph <- gsub(" ", "", cipher, fixed = TRUE)
blocks <- sub("\\s+$", "", gsub("(.{2})", "\\1 ",
   cph))
block1 <- substr(blocks, 1, 2)
p1 <- decrypt(block1)
block2 <- substr(blocks, 4, 5)
p2 <- decrypt(block2)
block3 <- substr(blocks, 7, 8)
```

```
p3 <- decrypt(block3)
secret_msg <- paste0(p1, p2, p3)
secret_msg <- sub("\\s+$", "", gsub("(.{3})", "\\1 ", secret_msg))
print(secret_msg)
```

R code Exa 10.3 example of cryptographic systems involving modular exponentiation

```
#page 204
library(gtools)
library(stringr)
encipher <- function(b, p) {
  two <- (b ^ 2) %% p
  four <- (two ^ 2) %% p
  eight <- (four ^ 2) %% p
  sixteen <- (eight ^ 2) %% p
  nineteen <- (b * two * sixteen) %% p
  return(nineteen)
}
message <- "SEND MONEY"
p <- 2609
k <- 19
char <- " "
plain_text <- gsub(" ", "[", message)
plain_text <- strsplit(plain_text, "")
plain_text_number <- vector()
encrypted_message <- vector()
for (ch in plain_text) {
 plain_text_number  <- append(plain_text_number, asc(ch) - 65)
}
block1 <- plain_text_number[1] * 100 + plain_text_number[2]
block2 <- plain_text_number[3] * 100 + plain_text_
```

```
    number[4]
block3 <- plain_text_number[5] * 100 + plain_text_
    number[6]
block4 <- plain_text_number[7] * 100 + plain_text_
    number[8]
block5 <- plain_text_number[9] * 100 + plain_text_
    number[10]
encrypted_message <- append(encrypted_message,
    encipher(block1, p))
encrypted_message <- append(encrypted_message,
    encipher(block2, p))
encrypted_message <- append(encrypted_message,
    encipher(block3, p))
encrypted_message <- append(encrypted_message,
    encipher(block4, p))
encrypted_message <- append(encrypted_message,
    encipher(block5, p))
for (i in seq_len(length(encrypted_message))) {
  encrypted_message[i]  <- str_pad(encrypted_message[
    i], 4, pad = "0")
  }
print(encrypted_message)
n <- round((1 - 4 * p) / k)
recovery_n <- (p - 1) + n
print(recovery_n)
```

R code Exa 10.4 an illustration of the RSA public key algorithm

```
#page 206
library(stringr)
library(gtools)
phi <- function(n) {
  for (num in 2 : sqrt(n))
    if (n %% num == 0) {
      p <- num
```

```
      q <- n / num
    }
  return((p - 1) * (q - 1))
}
encipher <- function(b, n) {
  two <- (b ^ 2) %% n
  four <- (two ^ 2) %% n
  eight <- (four ^ 2) %% n
  sixteen <- (eight ^ 2) %% n
  thirty_two <- (sixteen ^ 2) %% n
  forty_seven <- (b * two * four * eight * thirty_two) %% n
  return(forty_seven)
}
message <- "NO WAY TODAY"
n <- 2701
k <- 47
plain_text_number <- vector()
encrypted_message <- vector()
plaintext <- gsub(" ", "[", message, fixed = TRUE)
p_split <- strsplit(plaintext, "")
for (ch in p_split) {
  plain_text_number <- append(plain_text_number, asc(ch) - 65)
}
block1 <- plain_text_number[1] * 100 + plain_text_number[2]
block2 <- plain_text_number[3] * 100 + plain_text_number[4]
block3 <- plain_text_number[5] * 100 + plain_text_number[6]
block4 <- plain_text_number[7] * 100 + plain_text_number[8]
block5 <- plain_text_number[9] * 100 + plain_text_number[10]
block6 <- plain_text_number[11] * 100 + plain_text_number[12]
encrypted_message <- append(encrypted_message,
```

```
    encipher(block1, n))
encrypted_message <- append(encrypted_message,
    encipher(block2, n))
encrypted_message <- append(encrypted_message,
    encipher(block3, n))
encrypted_message <- append(encrypted_message,
    encipher(block4, n))
encrypted_message <- append(encrypted_message,
    encipher(block5, n))
encrypted_message <- append(encrypted_message,
    encipher(block6, n))
for (i in seq_len(length(encrypted_message))) {
  encrypted_message[i]  <- str_pad(encrypted_message
      [i], 4, pad = "0")
}
print(encrypted_message)
phi <- phi(n)
for (j in 2 : phi - 1) {
  if ((k * j) %% phi == 1) {
    return(j)
  }
}
print(j)
```

R code Exa 10.5 to solve the superincreasing knapsack problem

```
#page 210
lhs <- 28
co_x1 <- 3
co_x2 <- 5
co_x3 <- 11
co_x4 <- 20
co_x5 <- 41
ans <- vector()
if (co_x5 > lhs) {
```

```
  x5 <- 0
}
if (co_x4 < lhs) {
  if ((co_x1 + co_x2 + co_x3) < lhs) {
    x4 <- 1
    ans <- append(ans, co_x4)
  }
}
lhs <- lhs - (co_x5 * x5) - (co_x4 * x4)
if (co_x3 > lhs) {
  x3 <- 0
}
if (co_x2 < lhs) {
  if ((co_x1 + co_x2) == lhs) {
    ans <- append(ans, co_x1)
    ans <- append(ans, co_x2)
  }
}
ans <- sort(ans)
print(ans)
```

R code Exa 10.6 A public key cryptosystem based on the knapsack problem

```
#page 212
library(binaryLogic)
secret_key <- c(3, 5, 11, 20, 41)
m <- 85
a <- 44
mm <- vector()
cipher_text <- vector()
encrytion_key <- (secret_key * a) %% m
message <- "HELP US"
plain_text <- gsub(" ", "", message)
for (ch in plain_text) {
```

```
 mm <- append(mm, asc(ch) - 65)
}
mm <- as.binary(mm, size = 2, n = 5)
for (num in mm) {
  sum <- 0
  for (bit in 1 : 5) {
    sum <- sum + ((as.integer(num[bit])) * encrytion
        _key[bit])
  }
  cipher_text <- append(cipher_text, sum)
}
print(cipher_text)
```

R code Exa 10.7 to encrypt a message using knapsack

```
#page 213
library(binaryLogic)
library(gtools)
secret_key <- c(3, 5, 11, 20, 41, 83, 179, 344, 690,
    1042)
m <- 2618
a <- 929
count <- 0
digit <- 0
big_m <- vector()
block <- vector()
cipher_text <- vector()
encrytion_key <- (secret_key * a) %% m
message <- "NOT NOW"
plain_text <- gsub(" ", "", message)
for (ch in plain_text) {
  big_m <- append(big_m, asc(ch) - 65)
}
big_m <- as.binary(big_m, signed = FALSE,
   littleEndian = FALSE, size = 2, n = 5, logic =
```

```
    FALSE)
for (cond in big_m) {
  digit <- digit + 1
  for (n in 1:5) {
  if (digit %% 2) {
    if (cond[n]) {
      count <- count + encrytion_key[n]
    }
  } else {
      if (cond[n]) {
        count <- count + encrytion_key[n + 5]
      }
    if (n == 5) {
    print(count)
    count <- 0
    }
  }
  }
}
```

R code Exa 10.8 illustrate the selection of the public key

```
#page 215
p <- 113
r <- 3
k <- 37
two <- (r ^ 2) %% p
four <- (two ^ 2) %% p
eight <- (four ^ 2) %% p
sixteen <- (eight ^ 2) %% p
thirty_two <- (sixteen ^ 2) %% p
a <- (r * four * thirty_two) %% p
public_key <- c(p, r, a)
print(public_key)
```

R code Exa 10.9 to encrypt a message using ElGamal

```
#page 216
library(base)
message <- "SELL NOW"
k <- 15
public_key <- c(43, 3, 22)
p <- public_key[1]
r <- public_key[2]
a <- public_key[3]
j <- 23
m <- vector()
m_ <- ""
plain_text <- gsub(" ", "", message)
for (ch in plain_text) {
  m <- append(m, asc(ch) - 65)
}
two <- (r ^ 2) %% p
four <- (two ^ 2) %% p
eight <- (four ^ 2) %% p
sixteen <- (eight ^ 2) %% p
r_digit <- (r * two * four * sixteen) %% p
two <- (a ^ 2) %% p
four <- (two ^ 2) %% p
eight <- (four ^ 2) %% p
sixteen <- (eight ^ 2) %% p
digit <- (a * two * four * sixteen) %% p
for (b in m) {
  str <- (digit * b) %% p
  if (floor(str / 10) == 0) {
  m_ <- paste(m_, "0", toString(str),  sep = "")
  }else {
    m_ <- paste(m_, toString(str), sep = "")
  }
```

```
}
s <- substr(m_, 1, 2)
s1 <- paste0("(", r_digit, ",", s, ")")
s <- substr(m_, 3, 4)
s2 <- paste0("(", r_digit, ",", s, ")")
s <- substr(m_, 5, 6)
s3 <- paste0("(", r_digit, ",", s, ")")
s <- substr(m_, 7, 8)
s4 <- paste0("(", r_digit, ",", s, ")")
s <- substr(m_, 9, 10)
s5 <- paste0("(", r_digit, ",", s, ")")
s <- substr(m_, 11, 12)
s6 <- paste0("(", r_digit, ",", s, ")")
s <- substr(m_, 13, 14)
s7 <- paste0("(", r_digit, ",", s, ")")
cipher_text <- paste0(s1, s2, s3, s4, s5, s6, s7)
print(cipher_text)
```

R code Exa 10.10 Using ElGamal cryposystem to authenticate a received message

```
#page 217
p <- 43
r <- 3
a <- 22
k <- 15
b <- 13
j <- 25
message <- "SELL NOW"
c <- (r ^ j) %% p
digit <- (b - c * k) %% (p - 1)
for (d in 1 : 20) {
  if (((j * d) %% (p - 1)) == digit) {
    break ()
  }
```

```
}
ans <- c(c, d)
print(ans)
v1 <- ((a ^ c) %% p * (c ^ d) %% p) %% p
v2 <- (r ^ B) %% p
if (v1 == v2) {
  print("TRUE")
}
```

Chapter 13

REPRESENTATION OF INTEGERS AS SUMS OF SQUARES

R code Exa 13.1 to represent a positive integer as sum of two squares

```
#page 268
perfect_sq <- function(a) {
  sq <- sqrt(a)
  flr <- floor(sq)
  if ((sq - flr) == 0) {
    return(TRUE)
  }else {
    return(FALSE)
  }
}
n <- 54145
p <- vector()
k <- vector()
ans <- list()
equation <- list()
i <- 0
while (n %% 2 == 0) {
```

```
  i <- i + 1
  n <- n / 2
}
if (i != 0) {
  p <- append(p, 2)
  k <- append(k, i)
}
for (num in 3 : (n - 1)) {
  if (num %% 2 == 1) {
    i <- 0
    while (n %% num == 0) {
      i <- i + 1
      n <- n / num
    }
    if (i != 0) {
      p <- append(p, num)
      k <- append(k, i)
    }
  }
}
for (num in length(p)) {
 if (k[num] == 1) {
   if ((k[num] %% 4) == 1) {
     if (perfect_sq(p[num] - 1)) {
       square <- p[num] - 1
       equation <- append(equation, 1)
       equation <- append(equation, sqrt(square))
       ans <- append(list(ans), list(equation))
     }else if (perfect_sq(p[num] - 4)) {
         square <- p[num] - 4
         equation <- append(equation, 1)
         equation <- append(equation, sqrt(square))
         ans <- append(list(ans), list(equation))
       }
     }
 }else {
   ans <- append(ans, p[num])
 }
```

```
  }
print(ans)
```

R code Exa 13.2 to prove Lemma 2

```
#page 274
p <- 17
s1 <- vector()
s2 <- vector()
s1int <- vector()
s2int <- vector()
for (n in 0 : ((p - 1) / 2)) {
  s1 <- append(s1, ((1 + (n ^ 2))))
  s2 <- append(s2, (- (n ^ 2)))
}
s1int <- s1 %% 17
s2int <- s2 %% 17
for (x in s1int) {
  for (y in s2int) {
    if (x == 0 | x == 1) {
      next ()
    }
    if (y == 0 | y == 1) {
      next ()
    }
    if ((1 + (x ^ 2) %% p) == y) {
      x0 <- x
      y0 <- y
      return()
    }
    }
}
b <- which(s2int == y0)
y0 <- s2[b]
y <- sqrt(abs(y0))
```

```
x <- x0
print(x)
print(y)
k <- (1 + (x ^ 2) + (y ^ 2)) / p
print(k)
```

R code Exa 13.3 to write an integer as sum of four squares

```
#page 277
perf_sq <- function(i) {
  sqr <- sqrt(i)
  sqr_round <- round(sqrt(i))
  if ((sqr - sqr_round) == 0) {
    return(TRUE)
  } else {
    return(FALSE)
  }
}
n <- 459
sq <- vector()
for (i in 4 : n) {
  if (perf_sq(i)) {
    sq <- append(sq, sqrt(i))
  }
}
for (a in sq) {
  for (b in sq) {
    for (c in sq) {
      for (d in sq) {
        if (b >= a | c >= b | d >= c) {
          next ()
        }
        if ((a * a + b * b + c * c + d * d) == n) {
          x <- a
          y <- b
```

```
          z <- c
          w <- d
        }
      }
    }
  }
}
print(x)
print(y)
print(z)
print(w)
```

Chapter 15

CONTINUED FRACTIONS

R code Exa 15.3 solve a linear Diophantine equation

```
#page 316
library(MASS)
getfracs <- function(frac) {
  tmp <- strsplit(frac, "/")[[1]]
  list(num = as.numeric(tmp[1]), deno = as.numeric(
     tmp[2]))
}
convergents <- function(cf, p, q) {
  l <- length(cf)
  p <- append(p, cf[1])
  q <- append(q, 1)
  for (n in 2 : l) {
    s <- 0
    t <- n
    repeat {
      if (t == n | t == (n + 1)) {
        s <- as.fractions(s + (1 / cf[n]))
      } else {
      s <- (1 / s) + (1 / cf[n])
      }
      n <- n - 1
```

```
      if (n == 1) {
        break
      }
    }
    s <- (1 / s) + cf[1]
    s <- (as.fractions(s))
    s <- attr(s, "fracs")
    fracs <- getfracs(s)
    p <- append(p, fracs$num)
    q <- append(q, fracs$den)
  }
  print(p)
  q[2] <- 1
  print(q)
  x <- c * q[3]
  y <- (- c) * p[3]
  print(x)
  print(y)
}
eucli <- function(a, b) {
  cf <- vector()
  repeat {
  cf <- append(cf, floor(a / b))
  r <- a %% b
  if (r == 0) {
    break
  }
  a <- b
  b <- r
  }
  return(cf)
}
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
```

```
  if (x < 0) {
    return(- x)
  }else {
    return(x)
  }
}
p <- vector()
q <- vector()
a <- 172
b <- 20
c <- 1000
g <- gcd(a, b)
a <- a / g
b <- b / g
c <- c / g
cf <- eucli(a, b)
convergents(cf, p, q)
```

R code Exa 15.5 to find continued fraction expansion of a number

```
#page 326
n <- sqrt(23)
x <- vector()
a <- vector()
x[1] <- n
a[1] <- floor(x[1])
for (i in 2 : 10) {
x[i] <- 1 / (x[i - 1] - a[i -1])
a[i] <- floor(x[i])
}
print(a)
```

R code Exa 15.6 to find continued fraction expansion of a number

```
#page 327
n <- pi
x <- vector()
a <- vector()
x[1] <- n
a[1] <- floor(x[1])
for (i in 2 : 10) {
  x[i] <- 1 / (x[i - 1] - a[i - 1])
  a[i] <- floor(x[i])
}
print(a)
```

R code Exa 15.7 an example of illustrating the corollary to sought a fraction

```
#page 337
library(MASS)
library(fractional)
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(- x)
  }else {
    return(x)
  }
}
farey_seq <- function(i) {
  f <- vector()
  f[1] <- 0 / 1
  f[2] <- 1
  for (m in 2 : i) {
```

```
    f <- append(f, 1 / m)
    for (g in 2 : m) {
      if (gcd(g, m) == 1) {
        f <- append(f, g / m)
      }
    }
  }
  f <- sort(as.fractions(f))
  return(f)
}
n <- 5
x <- sqrt(7)
val <- x - 2
fn <- farey_seq(5)
for (k in seq_len(length(fn))) {
  if ((val > fn[k]) & (val < fn[k + 1])) {
    nu1 <- numerators(fn[k])
    d1 <- denominators(fn[k])
    nu2 <- nu1 + numerators(fn[k + 1])
    d2 <- d1 + denominators(fn[k + 1])
    if (nu2 / d2 > val) {
      u <- nu1
      v <- d1
    } else {
      u <- nu2 - nu1
      v <- d2 - d1
    }
  }
}
if (val - (u / v) < 1 / (v * (n + 1))) {
  ans <- as.fractions((u / v) + 2)
}
print(ans)
```

R code Exa 15.8 to solve an application of above theorem

```
#page 347
convergents <- function(cf) {
  l <- length(cf)
  ss <- vector()
  for (n in 2 : l) {
    s <- 0
    t <- n
    repeat {
      if (t == n) {
        s <- (s + (1 / cf[n]))
      } else {
        s <- 1 / (s + cf[n])
      }
      n <- n - 1
      if (n == 1) {
        break
      }
    }
    s <- s + cf[1]
    s <- fractional(s)
    ss <- append(ss, s)
  }
  return(ss)
}
cont_frac <- function(i) {
n <- sqrt(i)
x <- vector()
a <- vector()
x[1] <- n
a[1] <- floor(x[1])
for (k in 2 : 12) {
  x[k] <- 1 / (x[k - 1] - a[k - 1])
  a[k] <- floor(x[k])
}
return(a)
}
d <- 7
p <- vector()
```

```
q <- vector()
l <- vector()
cf <- cont_frac(d)
n <- 4
p <- append(p, cf[1])
q <- append(q, 1)
s <- convergents(cf)
for (j in 2 : length(s)) {
  p <- append(p, numerators(s[j]))
  q <- append(q, denominators(s[j]))
}
q[2] <- 1
if (n %% 2 == 0) {
  for (k in 1 : 3) {
    l <- append(l, (k * n) - 1)
  }
}else {
   for (k in 1: 3) {
     l <- append(l, (2 * k * n) - 1)
   }
 }
for (num in l) {
  print(p[num])
  print(q[num])
}
```

R code Exa 15.9 to find a solution of an equation for the smallest positive integer

```
#page 347
library(fractional)
convergents <- function(cf) {
  l <- length(cf)
  ss <- vector()
  for (n in 2 : l) {
```

```
    s <- 0
    t <- n
    repeat {
      if (t == n) {
        s <- (s + (1 / cf[n]))
      } else {
        s <- 1 / (s + cf[n])
      }
      n <- n - 1
      if (n == 1) {
        break
      }
    }
    s <- s + cf[1]
    s <- fractional(s)
    ss <- append(ss, s)
  }
  return(ss)
}
cont_frac <- function(i) {
  n <- sqrt(i)
  x <- vector()
  a <- vector()
  x[1] <- n
  a[1] <- floor(x[1])
  for (k in 2 : 10) {
    x[k] <- 1 / (x[k - 1] - a[k - 1])
    a[k] <- floor(x[k])
  }
  return(a)
}
d <- 13
p <- vector()
q <- vector()
cf <- cont_frac(d)
n <- 5
p <- append(p, cf[1])
q <- append(q, 1)
```

```
s <- convergents(cf)
for (j in 2 : length(s)) {
  p <- append(p, numerators(s[j]))
  q <- append(q, denominators(s[j]))
}
q[2] <- 1
k <- 1
if (n %% 2 == 0) {
    l <- (k * n) - 1
}else {
    l <- (2 * k * n) - 1
}
  print(p[l])
  print(q[l])
```

Chapter 16

SOME MODERN DEVELOPMENTS

R code Exa 16.1 factorization of a number using Pollards method

```
#page 359
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(- x)
  }else {
    return(x)
  }
}
f <- function(x) {
  return((x * x) - 1)
}
n <- 30623
x <- vector()
x[1] <- 3
```

```
for (k in 2 : 9) {
  x[k] <- f(x[k - 1]) %% n
}
for (k in seq_len(9 / 2)) {
 a <- x[2 * k] - x[k]
 g <- gcd(a, n)
 if (g != 1) {
   break
 }
}
p <- n / g
print(p)
print(g)
x <- x %% g
print(x)
```

R code Exa 16.2 to obtain a nontrivial divisor of a number

```
#page 361
library(gmp)
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(- x)
  }else {
    return(x)
  }
}
n <- 2987
a <- 2
q <- 7
```

```
s <- a
s <- as.bigz(s)
for (pow in 2 : q) {
  s <- (s ^ pow) %% n
}
s <- asNumeric(s)
ans <- gcd(s - 1, n)
print(ans)
```

R code Exa 16.3 to factor a number using the continued fraction factorization method

```
#page 362
library(fractional)
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(- x)
  }else {
    return(x)
  }
}
convergents <- function(cf) {
  l <- length(cf)
  ss <- vector()
  ss <- append(ss, cf[1])
  for (n in 2 : l) {
    s <- 0
    t <- n
    repeat {
      if (t == n) {
```

```
        s <- (s + (1 / cf[n]))
      } else {
        s <- 1 / (s + cf[n])
      }
      n <- n - 1
      if (n == 1) {
        break
      }
    }
    s <- s + cf[1]
    s <- fractional(s)
    ss <- append(ss, numerators(s))
  }
  return(ss)
}
cont_frac <- function(i) {
  n <- sqrt(i)
  x <- vector()
  a <- vector()
  x[1] <- n
  a[1] <- floor(x[1])
  for (k in 2 : 9) {
    x[k] <- 1 / (x[k - 1] - a[k - 1])
    a[k] <- floor(x[k])
  }
  return(a)
}
n <- 3427
s <- vector()
t <- vector()
a <- cont_frac(n)
p <- convergents(a)
s <- append(s, 0)
t <- append(t, 1)
for (num in seq_len(8)) {
  s[num + 1] <- (a[num] * t[num]) - s[num]
  t[num + 1] <- (n - (s[num + 1] ^ 2)) / t[num]
}
```

```
for (num in t) {
  if (num == 1) {
    next ()
  }
  sq <- sqrt(num)
  d <- round(sqrt(num))
  if (d == sq) {
    index <- num
    return()
  }
}
ans <- gcd(p[index - 1] + sqrt(t[index]), n)
ans2 <- gcd(p[index - 1] - sqrt(t[index]), n)
print(ans)
print(ans2)
```

R code Exa 16.4 to factor a number using the continued fraction factorization method

```
#page 363
library(fractional)
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(- x)
  }else {
    return(x)
  }
}
convergents <- function(cf) {
  l <- length(cf)
```

```
  ss <- vector()
  ss <- append(ss, cf[1])
  for (n in 2 : l) {
    s <- 0
    t <- n
    repeat {
      if (t == n) {
        s <- (s + (1 / cf[n]))
      } else {
        s <- 1 / (s + cf[n])
      }
      n <- n - 1
      if (n == 1) {
        break
      }
    }
    s <- s + cf[1]
    s <- fractional(s)
    ss <- append(ss, numerators(s))
  }
  return(ss)
}
cont_frac <- function(i) {
  n <- sqrt(i)
  x <- vector()
  a <- vector()
  x[1] <- n
  a[1] <- floor(x[1])
  for (k in 2 : 9) {
    x[k] <- 1 / (x[k - 1] - a[k - 1])
    a[k] <- floor(x[k])
  }
  return(a)
}
n <- 2059
s <- vector()
t <- vector()
a <- cont_frac(n)
```

```
p <- convergents(a)
s <- append(s, 0)
t <- append(t, 1)
for (num in seq_len(8)) {
  s[num + 1] <- (a[num] * t[num]) - s[num]
  t[num + 1] <- (n - (s[num + 1] ^ 2)) / t[num]
}
for (num in t) {
  for (num2 in t)
  if (num == 1 | num2 == 1 | num == num2) {
    next ()
  }
  sq <- sqrt(num * num2)
  d <- round(sqrt(num * num2))
  if (d == sq) {
    return()
  }
}
index <- match(num, t)
index2 <- match(num2, t)
x <- sqrt(t[index] * t[index2])
y <- (p[index - 1] * p[index2 - 1]) %% n
ans <- gcd(x + y, n)
ans2 <- n / ans
print(ans)
print(ans2)
```

R code Exa 16.5 an example of the quadratic sieve algorithm

```
#page 365
library(primes)
factorize <- function(n, f) {
  k <- vector()
  for (g in f) {
    i <- 0
```

```
    if (g == - 1) {
      if (n < 0) {
        n <- -1 * n
        k <- append(k, 1)
      } else {
        k <- append(k, 0)
      }
      next ()
    }
    while (n %% g == 0) {
      i <- i + 1
      n <- n / g
    }
    if (i != 0) {
      k <- append(k, i)
    } else {
      k <- append(k, 0)
    }
  }
  if (n == 1) {
  return(k)
  } else {
    return(66)
  }
}
fofx <- function(x) {
  return((x^2) - n)
}
check_residue <- function(a, p) {
  if (a == -1) {
    return(-1)
  }
  if (a > 1) {
    a <- a %% p
  }
  if (a == 1) {
    return(1)
  }
```

```
  if (a %% 2 == 0) {
    if (p %% 8 == 1 | p %% 8 == 7) {
    a <- a / 2
  } else {
    a <- (- 1 * a) / 2
  }
    return(check_residue(a, p))
  }
  if (a %% 2 != 0 && p %% 2 != 0) {
  if ((a %% 4 == 3) && (p %% 4 == 3)) {
    return(check_residue(- 1, a))
  } else {
    return(check_residue(p, a))
  }
    }
  return(0)
}
n <- 9487
kdata <- vector()
x <- floor(sqrt(n))
fb <- vector()
ex <- vector()
fb[1] <- -1
fb[2] <- 2
ap <- generate_primes(max = 30)
for (num in ap) {
  if (num == 2) {
    next ()
  }
  if (check_residue(n, num) == 1) {
    fb <- append(fb, num)
  }
}
f <- seq(x - 16, x + 16)
for (w in f) {
  k <- factorize(fofx(w), fb)
  if ( length(k) == 1) {
    ex <- append(ex, w)
```

```
    next ()
  }
  kdata <- c(kdata, k)
}
f <- f[!f %in% ex]
r <- length(fb)
c <- length(kdata) / r
p <- matrix(kdata, nrow = r, ncol = c, dimnames =
   list(fb, f))
for (i in seq_len(length(f))) {
  for (j in i : length(f)) {
    for (k in j : length(f)) {
      if (i == j | j == k | k == i) {
        next ()
      }
      for (h in seq_len(length(fb))) {
        m <- (p[h, i] + p[h, j] + p[h, k]) %% 2
        if (m != 0) {
          break
        } else if (h == length(fb)) {
          a <- i
          b <- j
          c <- k
          return()
        }
      }
    }
  }
}
lh <- (f[a] * f[b] * f[c]) %% n
sum <- 1
ma <- (p[, a] + p[, b] + p[, c])
for (h in seq_len(length(fb))) {
  if (ma[h] == 0) {
    next ()
  } else if (ma[h] == 2) {
    sum <- sum * fb[h]
  } else {
```

```
    sum <- sum * ((fb[h]) ^ (ma[h] - 2))
  }
}
if (sum < 0) {
sum <- -1 * sum
}
sum <- sum %% n
ans <- gcd(sum + lh, n)
print(ans)
ans2 <- n / ans
print(ans2)
```

R code Exa 16.6 Using a theorem to check if a number is prime

```
#page 367
mod <- function(a, b) {
  ans <- 1
  for (num in 1: b) {
    ans <- (ans * a) %% n
  }
  if (ans == n - 1) {
    ans <- -1
  }
  return(ans)
}
factorize <-  function(n) {
  number <- n
  p <- vector()
  i <- 0
  while ((n %% 2) == 0) {
    i <- i + 1
    n <- n / 2
  }
  if (i != 0) {
    p <- append(p, 2)
```

```
  }
  for (num in 3 : sqrt(number)) {
    if (num %% 2 == 1) {
      i <- 0
      while (n %% num == 0) {
        i <- i + 1
        n <- n / num
      }
      if (i != 0) {
        p <- append(p, num)
      }
    }
  }
  p <- append(p, n)
  return(p)
}
n <- 997
a <- 7
m <- vector()
modulus <- mod(a, n-1)
print(modulus)
p <- factorize(n-1)
for (num in p) {
  m <- append(m, mod(a, (n - 1) / num))
}
print(m)
```

R code Exa 16.7 to find four square roots of a modulo n

```
#page 372
library(primes)
solve_on <- function(a, b) {
  for (i in seq_len(10)) {
    c <- (1 - q * i) / p
    cr <- round(c)
```

```
    if (c == cr) {
      d <- i
      break
    }
  }
  x <- p * c * b + q * d * a
  return(x %% n)
}
a <- 324
n <- 391
ans <- vector()
for (h in generate_primes(max = sqrt(n))) {
  if (n %% h == 0) {
    p <- h
    q <- n / h
    break
  }
}
x1 <- a %% p
x2 <- a %% q
x2 <- sqrt(q + x2)
ans <- append(ans, solve_on(x1, -x2))
ans <- append(ans, solve_on(-x1, x2))
ans <- append(ans, solve_on(x1, x2))
ans <- append(ans, solve_on(- x1, - x2))
ans <- sort(ans)
print(ans)
```

R code Exa 16.8 to solve an example of blums game

```
#page 374
mod <- function(a, b, n) {
  ans <- 1
  for (num in 1: b) {
    ans <- (ans * a) %% n
```

```
  }
  if (ans == n - 1) {
    ans <- -1
  }
  return(ans)
}
solve_on <- function(a, b) {
  for (i in seq_len(20)) {
    c <- (1 - q * i) / p
    cr <- round(c)
    if (c == cr) {
      d <- i
      break
    }
  }
  x <- p * c * b + q * d * a
  return(x %% n)
}
gcd <- function(x, y) {
  while (y) {
    temp <- y
    y <- x %% y
    x <- temp
  }
  if (x < 0) {
    return(- x)
  }else {
    return(x)
  }
}
p <- 43
q <- 71
n <- p * q
s <- 192
ans <- vector()
a <- (s ^ 2) %% n
x1 <- a %% p
x2 <- a %% q
```

```
if (p %% 4 == 3 && q %% 4 == 3) {
  x1 <- mod(x1, ((p + 1) / 4), p)
  x2 <- mod(x2, ((q + 1) / 4), q)
}
x1 <- p - x1
x2 <- q - x2
ans <- append(ans, solve_on(x1, -x2))
ans <- append(ans, solve_on(-x1, x2))
ans <- append(ans, solve_on(x1, x2))
ans <- append(ans, solve_on(- x1, - x2))
ans <- sort(ans)
guess <- sample(ans, 1)
g1 <- gcd(s + guess, n)
g2 <- gcd(s - guess, n)
if (g1 == 1 && g2 == n) {
  print("Alice wins!")
} else {
  print("Bob wins!")
}
```