b) Código.

Primero definamos el sistema de ecuaciones:

```
F_x <- function(x) {
  c(
    x[1]^2 - 2*x[1] + x[2]^2 - x[3] + 1,
    x[1]*x[2]^2 - x[1] - 3*x[2] + x[2]*x[3] + 2,
    x[1]*x[3]^2 - 3*x[3] + x[2]*x[3]^2 + x[1]*x[2]
  )
}
```

La derivada vectorial del sistema anterior es:

$$\frac{\partial \phi(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} 2x_1 - 2 & 2x_2 & -1 \\ x_2^2 - 1 & 2x_1x_2 - 3 + x_3 & x_2 \\ x_3^2 + x_2 & x_3^2 + x_1 & 2x_1x_3 - 3 + 2x_2x_3 \end{pmatrix}$$

El siguiente código construye la derivada vectorial:

```
F_x_prime <- function(x) {
  matrix(c(
    c(2*x[1] - 2, 2*x[2], -1),
    c(x[2]^2 - 1, 2*x[1]*x[2] - 3 +x[3], x[2]),
    c(x[3]^2 + x[2], x[3]^2 + x[1], 2*x[1]*x[3] - 3 + 2*x[1]*x[2])
    ),
    nrow = 3,
    byrow = T)
}
```

Con la derivada vectorial podemos calcular la inversa; no obstante, como ya se comentó, analíticamente la inversa es muy grande, por lo que su cálculo sería demasiado laborioso. Por tanto, para la realización de las iteraciones primero evaluaremos la derivada vectorial en $\mathbf{x}_i$, resultado en una matriz numérica; posteriormente, calcularemos la inversa de esta matriz numérica, que será la utilizada en la aplicación del método de Newton.

En este sentido, el siguiente código realiza las iteraciones y muestra una matriz con los valores usados como condición inicial y sus sucesores, hasta la convergencia. Durante la aplicación se define una metodología para detener el algoritmo cuando la norma de la diferencia entre el vector de la iteración i y el vector de la iteración $i-1$ es menor a 5×10^{-11}. Esto asegura que la distancia total entre iteraciones consecutivas de la solución sea lo menor posible, indicando convergencia.

```
x0 <- c(1.5, 1.5, 1.5)
x_n <- matrix(ncol = 3)
x_n[1,] <- t(x0)

umbral <- 0.00000000005

for (i in 1:100) {

  F_x_prime_eval <- F_x_prime(x_n[i,])

  x_i <- x_n[i,] - solve(F_x_prime_eval)%*%F_x(x_n[i,])
```

```
  if (norm(t(x_i) - x_n[i,], type = "2")< umbral) {
    break
  }

  x_n <- rbind(x_n,t(x_i))

}

r1 <- x_n[nrow(x_n),]

x_n
```

```
##           [,1]     [,2]     [,3]
## [1,] 1.5000000 1.500000 1.500000
## [2,] 0.6901408 1.380282 1.330986
## [3,] 0.7903258 1.071115 1.085629
## [4,] 0.9584174 1.009240 0.988212
## [5,] 1.0004712 1.000542 0.999241
## [6,] 0.9999983 1.000000 1.000000
## [7,] 1.0000000 1.000000 1.000000
```

Mediante un proceso de prueba y error, se eligió a $\mathbf{x}_0 = \left(1.5, 1.5, 1.5\right)$ como condición inicial para hallar el primer vector de raíces

$$\mathbf{x}_1^* = \left(1, 1, 1\right)$$

Así, es posible observar que la aplicación del método de Newton converge después de 7 iteraciones.

En los sucesivo se muestra el código para hallar el segundo vector de raíces, utilizando como condición inicial $\mathbf{x}_0' = \left(0.5, 0.5, 0.5\right)$:

```
x0 <- c(0.5, 0.5, 0.5)
x_n <- matrix(ncol = 3)
x_n[1,] <- t(x0)

for (i in 1:100) {

  F_x_prime_eval <- F_x_prime(x_n[i,])

  x_i <- x_n[i,] - solve(F_x_prime_eval)%*%F_x(x_n[i,])

  if (norm(t(x_i) - x_n[i,], type = "2")< umbral) {
    break
  }

  x_n <- rbind(x_n,t(x_i))

}

r2 <- x_n[nrow(x_n),]
tail(x_n)
```

```
##           [,1]      [,2]      [,3]
## [53,] 1.098943 0.3676167 0.1449317
## [54,] 1.098943 0.3676167 0.1449317
## [55,] 1.098943 0.3676167 0.1449317
## [56,] 1.098943 0.3676167 0.1449317
## [57,] 1.098943 0.3676167 0.1449317
## [58,] 1.098943 0.3676167 0.1449317
```

Es decir, el segundo vector de raíces es

$$\mathbf{x}_2^* = \Big(1.0989426, 0.3676167, 0.1449317\Big)$$

Este resultado se obtiene después de 58 iteraciones.

Finalmente, existen librerías que aplican el método de Newton de manera muy simple. Uno de ellos es el de la librería pracma. A continuación los resultados de la función newtonsys:

```
r1_pracma <- pracma::newtonsys(F_x, x0 = c(1.5,1.5,1.5))
r2_pracma <- pracma::newtonsys(F_x, x0 = c(.5,.5,.5))

r1_pracma$zero
```

```
## [1] 1 1 1
```

```
r2_pracma$zero
```

```
## [1] 1.0989426 0.3676167 0.1449317
```

Los cuales son exactamente iguales a los hallados de forma manual.

iii) Código.

Primero generemos la muestra

```
rm(list = ls())
set.seed(1)

n <- 100
theta <- 1

x <- rlogis(n, location = theta, scale = 1)
```

Ahora generemos la función $\Phi(\theta|x_1, x_2, \dots, x_n)$:

```
phi <- function(theta) {

  n <- length(x)
  Sigma_xi <- 0

  for (i in 1:n) {
    Sigma_xi <- Sigma_xi + exp(theta - x[i]) / (1 + exp(theta - x[i]))
  }

  y <- Sigma_xi - n/2

  return(y)

}
```

Generemos también la derivada, i.e., $\Phi'(\theta|x_1, x_2, \dots, x_n)$

```
phi_prime <- function(theta) {

  n <- length(x)
  Sigma_xi <- 0

  for (i in 1:n) {
    Sigma_xi <- Sigma_xi + exp(theta - x[i]) / (1 + exp(theta - x[i]))^2
  }

  y <- Sigma_xi

  return(y)

}
```

Con lo anterior podemos aplicar el algoritmo, utilizando como condición inicial $\theta_0 = 0$. También detendremos el algoritmo una vez que la diferencia entre las soluciones de iteraciones consecutivas sea menor a 5×10^{-11}.

```
theta_0 <- 0
theta_n <- c()
theta_n[1] <- theta_0
```

```r
umbral <- 0.00000000005

for (i in 1:100) {

  theta_i <- theta_n[i] - phi(theta = theta_n[i])/phi_prime(theta = theta_n[i])

  if (abs(theta_i - theta_n[i]) < umbral) {
    break
  }

  theta_n[i+1] <- theta_i

}

iteraciones <- length(theta_n)
iteraciones
```

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

```r
theta_MV <- theta_n[iteraciones]
theta_MV
```

```
## [1] 1.099729
```

Es decir, tras 4 iteraciones, el método de Newton sugiere que el estimador de máxima verosimilitud es

$$\hat{\theta}_{MV}(x_1, x_2, \dots, x_{100}) = 1.0997287 \approx \theta = 1$$

El cual es muy cercano al valor verdadero de θ.

Podemos corroborar lo anterior utilizando la función `newton` de la librería `pracma`:

```r
pracma_theta_MV <- pracma::newton(phi, x0 = theta_0)
pracma_theta_MV$niter
```

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

```r
pracma_theta_MV$root
```

```
## [1] 1.099729
```

Los resultados son muy similares que los arrojados por el algoritmo construido desde cero.