Ejercicios de Fundamentos de Programación

Grado en Biotecnología

Diciembre 2024

POLITÉCNICA

Autora: Virginia Yagüe Jiménez

Ejercicios básicos de R

Los siguientes ejercicios son de tipo básico. No emplean uso de funciones pero sirven de ejemplo de como implementar código R que calcule ciertos modelos útiles en el contexto de la biotecnología.

Las soluciones se encuentran al final del capítulo. Recuerda que es posible que alguno de los ejercicios se puedan resolver de forma alternativa a la propuesta en este documento.

Ejercicio 1: Crecimiento celular acumulativo

En biología, el crecimiento celular en un entorno dinámico puede modelarse considerando factores de crecimiento y aportes externos de nutrientes en cada unidad de tiempo. Un modelo matemático sencillo para describir este fenómeno es:

$$P(t) = P_0 \prod_{i=1}^{t} (1 + r_i) + \sum_{i=1}^{t} F_i$$

Descripción del modelo:

- $P(t)$: Población celular en el tiempo t.
- P_0: Población inicial de células.
- r_i: Tasa de crecimiento relativa en el intervalo i, donde $i = 1, 2, \ldots, t$.
- F_i: Cambio neto en la población celular debido a factores externos como aportes de nutrientes o pérdidas.

Supongamos que una colonia celular inicia con una población de $P_0 = 500$ células. Durante los siguientes cinco períodos, la tasa de crecimiento y los aportes externos varían de la siguiente manera:

- **Tasa de crecimiento relativa en cada período (r):** $r_1 = 0.01$, $r_2 = 0.02$, $r_3 = 0.03$, $r_4 = 0.01$, $r_5 = 0.04$
- **Aportes externos en cada período (F):** $F_1 = 10$, $F_2 = 15$, $F_3 = -5$, $F_4 = 20$, $F_5 = 10$

Calcula la población celular al final del período $t = 5$, es decir, $P(5)$.

Ejercicio 2: Transporte de Oxígeno en el Sistema Circulatorio

El transporte de oxígeno en el sistema circulatorio se puede modelar matemáticamente teniendo en cuenta la eficiencia de transporte en diferentes etapas y la dispersión del oxígeno durante el proceso. Un modelo simplificado para representar este fenómeno es:

$$Q = \prod_{i=1}^{n} s_i + \sum_{i=1}^{n} \frac{d_i}{s_i}$$

Las variables involucradas en el modelo son:

- Q: Cantidad total de oxígeno transportada.
- s_i: Eficiencia de transporte en la etapa i.
- d_i: Dispersión o pérdida de oxígeno en la etapa i.
- n: Número total de etapas en el proceso de transporte.

Considera un sistema circulatorio simplificado que transporta oxígeno a través de tres etapas. Las eficiencias de transporte y las dispersión de oxígeno en cada etapa están definidas de la siguiente manera:

- **Eficiencias de transporte (s):** $s_1 = 0.9$, $s_2 = 0.8$, $s_3 = 0.85$
- **Dispersión de oxígeno (d):** $d_1 = 0.5$, $d_2 = 0.4$, $d_3 = 0.6$

Calcula la cantidad total de oxígeno transportada (Q) aplicando el modelo descrito.

Ejercicio 3: Flujo metabólico

En el estudio del metabolismo celular, es fundamental analizar cómo los factores externos e internos influyen en el flujo metabólico a lo largo de distintas etapas. Este flujo se puede modelar mediante la siguiente ecuación:

$$F = \sum_{i=1}^{n} k_i \prod_{j=1}^{i} c_j$$

Descripción del modelo:

- F: Flujo metabólico acumulado.
- k_i: Coeficiente de contribución al flujo en la etapa i.
- c_j: Factor multiplicativo en la etapa j.

- n: Número total de etapas del proceso.

Considera un sistema metabólico que evoluciona a lo largo de tres etapas sucesivas. Los coeficientes y factores relacionados con el flujo metabólico están definidos de la siguiente manera:

- **Coeficientes de flujo (k):** $k_1 = 0.1$, $k_2 = 0.2$, $k_3 = 0.15$
- **Factores acumulativos (c):** $c_1 = 1.0$, $c_2 = 1.2$, $c_3 = 1.1$

Soluciones:

Solución Ejercicio 1: Crecimiento celular acumulativo

Sin usar `prod` y `sum`

```r
P0 <- 500
r <- c(0.01, 0.02, 0.03, 0.01, 0.04)
F <- c(10, 15, -5, 20, 10)

productorio <- 1
for (i in 1:length(r)) {
  productorio <- productorio * (1 + r[i])
}

sumatorio <- 0
for (i in 1:length(F)) {
  sumatorio <- sumatorio + F[i]
}

P_t <- P0 * productorio + sumatorio
cat("Población total después de 5 días:", P_t, "\n")
```

Usando `prod` y `sum`

```r
P0 <- 500
r <- c(0.01, 0.02, 0.03, 0.01, 0.04)
F <- c(10, 15, -5, 20, 10)

productorio <- prod(1 + r)
sumatorio <- sum(F)
P_t <- P0 * productorio + sumatorio
cat("Población total después de 5 días:", P_t, "\n")
```

Solución Ejercicio 2: Transporte de oxígeno

Sin usar `prod` y `sum`

```r
s <- c(0.9, 0.8, 0.85)
d <- c(0.5, 0.4, 0.6)

productorio <- 1
for (i in 1:length(s)) {
  productorio <- productorio * s[i]
}

sumatorio <- 0
for (i in 1:length(d)) {
  sumatorio <- sumatorio + (d[i] / s[i])
}

Q <- productorio + sumatorio
cat("Transporte total de oxígeno (Q):", Q, "\n")
```

Usando `prod` y `sum`

```r
s <- c(0.9, 0.8, 0.85)
d <- c(0.5, 0.4, 0.6)

productorio <- prod(s)
sumatorio <- sum(d / s)
Q <- productorio + sumatorio
cat("Transporte total de oxígeno (Q):", Q, "\n")
```

Solución Ejercicio 3: Flujo metabólico

Sin usar `prod` y `sum`

```r
k <- c(0.1, 0.2, 0.15)
c <- c(1.0, 1.2, 1.1)

F <- 0
for (i in 1:length(k)) {
  producto <- 1
  for (j in 1:i) {
    producto <- producto * c[j]
  }
  F <- F + k[i] * producto
}

cat("Flujo metabólico total (F):", F, "\n")
```

Usando `prod` y `sum`

```r
k <- c(0.1, 0.2, 0.15)
c <- c(1.0, 1.2, 1.1)

F <- sum(sapply(1:length(k), function(i) k[i] * prod(c[1:i])))
cat("Flujo metabólico total (F):", F, "\n")
```

Bucles y Condiciones

Los ejercicio de este capítulo son algo más complejos que los recogidos en el anterior, ya que requieren estructuras como `while`, `if`, y `else`.

Las soluciones se encuentran al final del capítulo. Recuerda que es posible que alguno de los ejercicios se puedan resolver de forma alternativa a la propuesta en este documento.

Ejercicio 1: Crecimiento Celular con Umbral

El crecimiento celular puede variar dependiendo de condiciones externas, como la cantidad de nutrientes disponibles. Este modelo considera un crecimiento limitado por un umbral máximo, donde si la población excede cierto valor, el crecimiento se detiene.

$$P(t) = P_0 \prod_{i=1}^{t} (1 + r_i) + \sum_{i=1}^{t} F_i$$

Descripción del modelo:

- $P(t)$: Población celular en el tiempo t.
- P_0: Población inicial de células.
- r_i: Tasa de crecimiento relativa en el intervalo i, donde $i = 1, 2, \ldots, t$.
- F_i: Cambio neto en la población celular debido a factores externos.

Supón que:

- $P_0 = 500$
- $r = \{0.01, 0.02, 0.03, 0.01, 0.04\}$
- $F = \{10, 15, -5, 20, 10\}$
- Umbral máximo: $P_{\text{max}} = 600$

Calcula la población celular $P(t)$ hasta que el valor supere P_{max} o finalice el ciclo.

Ejercicio 2: Transporte de Oxígeno con Pérdidas Críticas

En un sistema circulatorio, el transporte de oxígeno puede verse comprometido si las pérdidas de oxígeno superan un valor crítico.

$$Q = \prod_{i=1}^{n} s_i + \sum_{i=1}^{n} \frac{d_i}{s_i}$$

Descripción del modelo:

- Q: Cantidad total de oxígeno transportada.
- s_i: Eficiencia de transporte en la etapa i.
- d_i: Pérdida de oxígeno en la etapa i.

Dado:

- $s = \{0.9, 0.8, 0.85\}$
- $d = \{0.5, 0.4, 0.6\}$
- Límite crítico de pérdida: $D_{\text{crit}} = 1.0$

Calcula Q, pero interrumpe el cálculo si la suma acumulada de d_i/s_i excede D_{crit}.

Ejercicio 3: Flujo Metabólico Condicionado

El flujo metabólico puede depender de ciertos factores que afectan las etapas intermedias. Si en alguna etapa el producto acumulado supera un valor crítico, se aplica un coeficiente reductor.

$$F = \sum_{i=1}^{n} k_i \prod_{j=1}^{i} c_j$$

Descripción del modelo:

- F: Flujo metabólico acumulado.
- k_i: Coeficiente de contribución al flujo en la etapa i.
- c_j: Factor multiplicativo en la etapa j.

Dado:

- $k = \{0.1, 0.2, 0.15\}$

- $c = \{1.0, 1.2, 1.1\}$
- Valor crítico: $C_{\text{crit}} = 1.5$
- Coeficiente reductor: $R = 0.9$

Calcula F, aplicando R si en alguna etapa el producto acumulado excede C_{crit}.

Ejercicio 4: Modelado de Tasa de Degradación con Cambios Dinámicos

En sistemas biológicos, las tasas de degradación de sustancias pueden variar dependiendo de factores ambientales como la temperatura o la presencia de otros compuestos. En este ejercicio, implementaremos un modelo dinámico de la tasa de degradación de una sustancia en función del tiempo, donde la tasa de degradación cambia bajo ciertas condiciones.

El modelo se describe mediante la siguiente ecuación:

$$D(t) = D_0 \left(1 - \frac{\alpha_i}{\beta_i}\right)^t$$

Donde el modelo contempla las siguientes variables:

- $D(t)$: Nivel de sustancia presente después de t ciclos de degradación.
- $D_0 = 1000$: Nivel inicial de sustancia (en unidades arbitrarias).
- α_i: Factor de degradación para el i-ésimo ciclo. Este factor depende de las condiciones ambientales y puede aumentar si el nivel de la sustancia $D(t)$ cae por debajo de un umbral.
- β_i: Factor de estabilidad, que refleja la resistencia de la sustancia a la degradación bajo condiciones normales.
- Los valores iniciales de α_i y β_i están definidos como vectores con diferentes valores para cada ciclo de degradación.

El nivel de sustancia $D(t)$ se evalúa en cada ciclo:

- Si $D(t) \leq 500$, entonces el factor de degradación α_i de cada ciclo se incrementa en un 20%.
- Si en cualquier ciclo el valor de α_i es mayor que β_i, el proceso de degradación se detiene inmediatamente, ya que la degradación sería excesiva para el sistema.

Implementa el modelo en R y realiza una simulación donde se calcule el nivel de sustancia $D(t)$ en cada ciclo de degradación. El proceso debe detenerse si alguna de las condiciones de terminación (cuando $\alpha_i > \beta_i$) se cumple, y el programa debe imprimir el nivel de la sustancia en cada iteración. Verifica el código para los siguientes datos:

- $D_0 = 1000$
- Factores de degradación iniciales: $\alpha = [0.2, 0.15, 0.12, 0.1, 0.08]$
- Factores de estabilidad: $\beta = [0.5, 0.45, 0.4, 0.35, 0.3]$

Ejercicio 5: Control de Producción en Cadena Biológica

En una cadena de producción biológica, si la suma acumulada de los productos excede cierto umbral, el sistema reduce la producción por factores externos. Si dos etapas consecutivas muestran una disminución en los valores, el sistema detiene la producción.

$$P_t = \sum_{i=1}^{n} (k_i \cdot p_i)$$

En donde:

- P_t: Es la producción total acumulada en el tiempo t.
- k_i: Es el coeficiente de eficiencia.
- p_i: Es la producción en cada etapa.

Implementa el modelo en R y realiza una simulación de la producción total P_t a lo largo de las etapas. En cada etapa, evalúa si las condiciones para reducir la eficiencia o detener la producción se cumplen. Si la producción acumulada supera el umbral de 800, ajusta el coeficiente de eficiencia. Si dos etapas consecutivas muestran una disminución en la producción, detén el proceso y muestra el mensaje correspondiente.

Verifica el programa para los siguientes datos:

- Coeficientes de eficiencia: $k = [1.0, 0.9, 0.85, 0.8, 0.75]$
- Producción en cada etapa: $p = [200, 180, 170, 160, 150]$
- El sistema comienza con una producción acumulada $P_t = 0$.

Ejercicio 6: Modelado de Fluctuaciones de Energía con Reinicio Condicional

El flujo de energía en un sistema puede fluctuar debido a cambios externos. Si la energía cae por debajo de un umbral, se reinicia a un valor inicial y el proceso continúa. El modeo por tanto se puede describir mediante la siguiente expresión:

$$E(t) = E_0 + \sum_{i=1}^{t} \Delta E_i$$

En donde:

- $E(t)$: Energía acumulada en el tiempo t.
- $E_0 = 100$: Energía inicial.
- ΔE_i: Cambio en la energía en cada paso.

Sin embargo, el modelo ha de cumplir las siguientes condiciones por iteración:

- Reiniciar $E(t)$ a 100 si cae por debajo de 50.
- Terminar si $E(t) > 500$ en algún momento.

Verifica el código para los siguientes datos:

- $E_0 = 100$
- Cambios de energía en cada etapa: $\Delta E_i = \{20, -30, 15, -50, 40, -10\}$

Soluciones

Solución Ejercicio 1: Crecimiento Celular con Umbral

```r
P0 <- 500
r <- c(0.01, 0.02, 0.03, 0.01, 0.04)
F <- c(10, 15, -5, 20, 10)
P_max <- 600

P_t <- P0
i <- 1
while (i <= length(r) && P_t <= P_max) {
  P_t <- P_t * (1 + r[i]) + F[i]
  i <- i + 1
}
cat("Población final:", P_t, "\n")
```

Solución Ejercicio 2: Transporte de Oxígeno con Pérdidas Críticas

```r
s <- c(0.9, 0.8, 0.85)
d <- c(0.5, 0.4, 0.6)
D_crit <- 1.0

Q <- 1
sum_d <- 0
i <- 1
while (i <= length(s)) {
  Q <- Q * s[i]
  sum_d <- sum_d + d[i] / s[i]
  if (sum_d > D_crit) {
    cat("Pérdidas críticas alcanzadas. Interrumpiendo cálculo.\n")
    break
  }
  i <- i + 1
}
cat("Oxígeno transportado (Q):", Q + sum_d, "\n")
```

Solución Ejercicio 3: Flujo Metabólico Condicionado

```r
k <- c(0.1, 0.2, 0.15)
c <- c(1.0, 1.2, 1.1)
C_crit <- 1.5
R <- 0.9

F <- 0
for (i in 1:length(k)) {
  producto <- 1
  for (j in 1:i) {
    producto <- producto * c[j]
  }
  if (producto > C_crit) {
    producto <- producto * R
  }
  F <- F + k[i] * producto
}
cat("Flujo metabólico total (F):", F, "\n")
```

Solución Ejercicio 4: Modelado de Tasa de Degradación con Cambios Dinámicos

```r
D0 <- 1000
alpha <- c(0.2, 0.15, 0.12, 0.1, 0.08)
beta <- c(0.5, 0.45, 0.4, 0.35, 0.3)
D_t <- D0

for (i in 1:length(alpha)) {
  if (D_t <= 500) {
    alpha[i] <- alpha[i] * 1.2
  }
  if (alpha[i] > beta[i]) {
    cat("Degradación detenida en iteración:", i, "\n")
    break
  }
  D_t <- D_t * (1 - alpha[i] / beta[i])
  cat("Nivel de sustancia en iteración", i, ":", D_t, "\n")
}
```

Solución Ejercicio 5: Control de Producción en Cadena Biológica

```r
k <- c(1.0, 0.9, 0.85, 0.8, 0.75)
p <- c(200, 180, 170, 160, 150)
P_t <- 0
consecutive_decrease <- 0

for (i in 1:length(k)) {
  P_t <- P_t + k[i] * p[i]
  if (P_t > 800) {
    k[i] <- k[i] * 0.7
  }
  if (i > 1 && p[i] < p[i - 1]) {
    consecutive_decrease <- consecutive_decrease + 1
```

```
  } else {
    consecutive_decrease <- 0
  }
  if (consecutive_decrease == 2) {
    cat("Producción detenida por disminución consecutiva.\n")
    break
  }
  cat("Producción acumulada:", P_t, "\n")
}
```

Solución Ejercicio 6: Modelado de Fluctuaciones de Energía con Reinicio Condicional

```
E0 <- 100
delta_E <- c(20, -30, 15, -50, 40, -10)
E_t <- E0

for (i in 1:length(delta_E)) {
  E_t <- E_t + delta_E[i]
  if (E_t < 50) {
    E_t <- 100
    cat("Energía reiniciada a 100 en iteración", i, "\n")
  }
  if (E_t > 500) {
    cat("Energía supera el umbral de 500. Proceso finalizado.\n")
    break
  }
  cat("Energía acumulada en iteración", i, ":", E_t, "\n")
}
```

Ejercicios con control de bucles usando `seq_len`, `seq_along` y `1:n`

En R, los bucles `for` pueden controlarse de manera eficiente usando `seq_len`, `seq_along` y `1:n`. Estas funciones permiten iterar de manera segura sobre vectores, garantizando la correcta definición de índices y evitando errores.

- **`seq_len(n)`**: Crea una secuencia de enteros desde 1 hasta n.
- **`seq_along(x)`**: Genera una secuencia desde 1 hasta la longitud del vector `x`.
- **`1:n`**: Genera una secuencia de enteros de 1 a n, pero puede fallar si $n = 0$, lo que hace más seguras `seq_len` y `seq_along`.

A continuación, presentamos ejercicios prácticos que emplean estas estructuras.

Ejercicio 1: Análisis de temperatura diaria

Supón que tienes un registro de temperaturas diarias en una ciudad durante 7 días. El modelo para calcular la temperatura promedio durante estos días es:

$$T_{\text{prom}} = \frac{1}{n} \sum_{i=1}^{n} T_i$$

Donde:

- T_i: Temperatura en el día i.
- n: Número de días.

Dado el siguiente conjunto de temperaturas:

$$T = \{23, 25, 27, 26, 24, 23, 22\}$$

Calcula la temperatura promedio.

Ejercicio 2: Ajuste de concentración de reactivos

Un experimento de química requiere ajustar la concentración de varios reactivos en 5 pasos. Cada paso i tiene una concentración inicial C_i y una variación diaria d_i. La concentración ajustada después de n días se calcula como:

$$C_n = C_1 \cdot \prod_{i=2}^{n} (1 + d_i)$$

Donde:

- C_1: Concentración inicial del reactivo.
- d_i: Variación de la concentración en el día i.
- n: Número de días.

Dado:

$$C_1 = 10, \quad d = \{0.02, 0.03, 0.01, -0.02, 0.05\}$$

Calcula la concentración final después de los 5 días.

Ejercicio 3: Evolución de la población de una especie

En un ecosistema, la población de una especie se ajusta día a día según la fórmula:

$$P_{i+1} = P_i \cdot (1 + r_i) - m_i$$

Donde:

- P_i: Población en el día i.
- r_i: Tasa de crecimiento de la población en el día i.
- m_i: Mortalidad en el día i.

Dado:

$$P_0 = 500, \quad r = \{0.05, 0.04, 0.03, 0.02, 0.01\}, \quad m = \{10, 15, 12, 8, 5\}$$

Calcula la población final después de 5 días.

Soluciones:

Solución Ejercicio 1

```r
T <- c(23, 25, 27, 26, 24, 23, 22)
n <- length(T)
T_prom <- sum(T) / n
cat("Temperatura promedio:", T_prom, "\n")
```

Solución Ejercicio 2

```r
C1 <- 10
d <- c(0.02, 0.03, 0.01, -0.02, 0.05)
C_final <- C1
for (i in seq_len(length(d))) {
# la linea anterior se puede sustituir por for (i in seq_along(d)) {
  C_final <- C_final * (1 + d[i])
}
cat("Concentración final:", C_final, "\n")
```

Solución Ejercicio 3

```r
P <- 500
r <- c(0.05, 0.04, 0.03, 0.02, 0.01)
m <- c(10, 15, 12, 8, 5)

for (i in seq_len(length(r))) {
# la linea anterior se puede sustituir por for (i in seq_along(r)) {
  P <- P * (1 + r[i]) - m[i]
}
cat("Población final:", P, "\n")
```

Funciones y estructuras de control

Los siguientes ejercicios están diseñados para profundizar en el uso de funciones en R, utilizando estructuras de control como `if-else`, `while` y `for`. Cada ejercicio se resuelve mediante funciones definidas por el usuario y evita el uso de funciones predefinidas avanzadas.

Las soluciones se encuentran al final del capítulo. Recuerda que es posible que alguno de los ejercicios se puedan resolver de forma alternativa a la propuesta en este documento.

Ejercicio 1: Serie Condicional

Define una función que calcule la suma de los primeros n términos de la serie:

$$S = \sum_{i=1}^{n} \begin{cases} 2i, & \text{si } i \text{ es par} \\ -i, & \text{si } i \text{ es impar} \end{cases}$$

Verifica la función para $n = 10$.

Ejercicio 2: Producto Acumulado hasta un Umbral

Define una función que calcule el producto acumulado de una secuencia de números $\{1, 2, 3, \ldots, n\}$ hasta que el producto exceda un valor umbral T. Usa un bucle `while`.

Verifica la función para $n = 10$ y $T = 100$.

Ejercicio 3: Evaluación de Función Polinómica

Define una función que evalúe un polinomio de la forma:

$$P(x) = \sum_{i=0}^{n} a_i x^i$$

Usando un bucle `for` y `if-else`, implementa la evaluación para $P(x)$ con coeficientes dados y un valor específico de x.

Verifica la función para $P(x) = 1 - 2x + 3x^2$ en el punto $x = 2$.

Ejercicio 4: Aproximación de Raíz Cuadrada por Iteración

Define una función que estime la raíz cuadrada de un número N usando el método de Herón. El método de Herón para calcular la raíz cuadrada de un número N es un algoritmo iterativo que comienza con una estimación inicial x_0 de la raíz cuadrada de N, y luego mejora esta estimación mediante la siguiente fórmula iterativa:

$$x_{k+1} = \frac{x_k + \frac{N}{x_k}}{2}$$

Donde:

- x_k es la estimación en la k-ésima iteración,
- x_{k+1} es la nueva estimación en la iteración siguiente,
- N es el número del que queremos calcular la raíz cuadrada.

El algoritmo paso a paso es el siguiente:

1. **Paso inicial**: Comienza con una estimación inicial x_0, que puede ser cualquier número positivo. Generalmente, se elige un valor como $x_0 = N/2$, ya que es una suposición razonable para la mayoría de los valores de N.
2. **Iteración**: A partir de x_0, usamos la fórmula iterativa para obtener la siguiente estimación x_1:

   $$x_1 = \frac{x_0 + \frac{N}{x_0}}{2}$$

   Esta fórmula ajusta la estimación x_0 de manera que se acerque más a la raíz cuadrada verdadera de N.
3. **Repetición**: Continuamos aplicando la misma fórmula iterativa:

   $$x_{k+1} = \frac{x_k + \frac{N}{x_k}}{2}$$

   con cada nuevo valor de x_k, hasta que la diferencia entre dos valores sucesivos x_k y x_{k+1} sea lo suficientemente pequeña, es decir, cuando:

   $$|x_{k+1} - x_k| < \epsilon$$

   donde ϵ es un valor pequeño que define la tolerancia de la aproximación.
4. **Resultado**: El proceso continúa hasta que se alcanza la precisión deseada y el último valor x_{k+1} será nuestra aproximación de la raíz cuadrada de N.

Define la función de tal forma que por defecto tome un valor de $\epsilon = 0.0001$, pero que este valor pueda ser modificado a la hora de invocar la función.

Verifica la función para $N = 25$.

Ejercicio 5: Serie Fibonacci Condicional

Define una función que calcule los primeros n términos de la serie de Fibonacci modificada, donde:

$$F(i) = \begin{cases} F(i-1) + F(i-2) & \text{si } i \text{ es par} \\ F(i-1) - F(i-2) & \text{si } i \text{ es impar} \end{cases}$$

La función debe usar una función interna para calcular los términos. Recuerda que la serie de Fibonacci toma estos valores iniciales $F(1) = 0$ y $F(2) = 1$.

Verifica la función para $n = 10$.

Ejercicio 6: Estimación de e^x mediante Serie de Taylor

Escribe una función que estime el valor de e^x usando la expansión en serie de Taylor:

$$e^x = \sum_{k=0}^{n} \frac{x^k}{k!}$$

Crea una función interna para calcular $k!$ (factorial) y otra para la potencia.

Ejercicio 7: Integración Numérica por Sumas de Riemann

Desarrolla una función en R que realice la integración numérica aproximada de una función $f(x)$ definida por el usuario mediante el método de **Sumas de Riemann** usando puntos medios.

Instrucciones:

1. La función debe aceptar como argumentos:
   - f: la función a integrar.
   - a: el límite inferior de integración.
   - b: el límite superior de integración.
   - n: el número de subintervalos en los que se dividirá $[a, b]$.
2. La función calculará la integral definida como:

   $$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} f(x_i)\Delta x$$

   donde:
   - $\Delta x = \frac{b-a}{n}$ es el ancho de cada subintervalo.
   - $x_i = a + (i - 0.5)\Delta x$ es el punto medio de cada subintervalo.
3. Implementa la solución usando un bucle `for` y evita el uso de funciones predefinidas avanzadas como `sum` y `prod`.
4. Incluye una función interna dentro de la función principal que evalúe los puntos medios x_i de cada subintervalo.

Fórmulas a emplear:

- **Ancho del subintervalo:**

$$\Delta x = \frac{b-a}{n}$$

- **Punto medio del subintervalo i:**

$$x_i = a + (i - 0.5)\Delta x$$

- **Suma acumulada:**

$$\text{suma} = \sum_{i=1}^{n} f(x_i)\Delta x$$

Verifica el código para $f(x)) = x^2$, $a = 0$, $b = 1$ y $n = 1000$

Ejercicio 8: Método de Bisección para Raíces

Define una función que encuentre la raíz de una función continua $f(x)$ en un intervalo $[a, b]$ usando el método de bisección. La función debe incluir verificaciones y funciones internas para evaluar $f(x)$.

1. La función debe aceptar como parámetros:
   - f: la función a la que se le buscará la raíz.
   - a: el límite inferior del intervalo.
   - b: el límite superior del intervalo.
   - ϵ: el criterio de tolerancia, es decir, la diferencia entre dos iteraciones consecutivas de la raíz encontrada.
   - max_iter: el número máximo de iteraciones permitidas.
2. La función debe verificar que $f(a)$ y $f(b)$ tengan signos opuestos. Si no los tienen, debe devolver un mensaje de error indicando que no se garantiza que exista una raíz en el intervalo.
3. Implementa la búsqueda de la raíz mediante el siguiente algoritmo:
   - Calcula el punto medio $m = \frac{a+b}{2}$.
   - Si $f(m)$ es suficientemente cercano a cero (según el valor de ϵ), entonces m es la raíz aproximada.
   - Si el signo de $f(a)$ y $f(m)$ son opuestos, la raíz está en el intervalo $[a, m]$. Actualiza el valor de $b = m$.
   - Si el signo de $f(b)$ y $f(m)$ son opuestos, la raíz está en el intervalo $[m, b]$. Actualiza el valor de $a = m$.
   - Repite el proceso hasta que la diferencia entre a y b sea menor que ϵ o se alcance el número máximo de iteraciones.

Al finalizar las iteraciones, la función debe devolver el valor de la raíz aproximada. Usa una función interna para evaluar $f(x)$ y otra para verificar las condiciones del intervalo.

Verifica el funcionamiento de tu código para $f(x) = x^3 - 4$ en el intervalo $[1, 2]$ y una tolerancia $\epsilon = 0.001$ y un numero máximo de iteraciones de 100.

Ejercicio 9: Determinación de Números Primos

Define una función que determine si un número n es primo. Utiliza un bucle `for` para verificar si n tiene divisores distintos de 1 y de sí mismo que le precedan.

Verifica el código para $n = 17$ y para $n = 18$

Ejercicio 10: Cálculo del Máximo Común Divisor (MCD)

Define una función que calcule el Máximo Común Divisor (MCD) de dos números a y b utilizando el algoritmo de Euclides. Éste es un método eficiente y antiguo para calcular el Máximo Común Divisor (MCD) de dos números enteros. Este algoritmo se basa en un principio simple pero poderoso: el MCD de dos números a y b es igual al MCD del número menor b y el resto de la división del mayor a entre el menor b.

Algoritmo de Euclides (MCD)

función MCD(a, b)
mientras b distinto de 0
r = a mod b
a = b
b = r
retornar a

Verifica el código para $a = 56$ y $b = 98$.

Ejercicio 11: Generación de una Tabla de Multiplicar

Define una función que genere la tabla de multiplicar de un número n hasta un límite m. La tabla debe ser impresa por pantalla tal y como se muestra en esta salida de ejemplo donde $n = 3$ y $m = 10$:

```
Tabla de multiplicar de 3:
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30
```

Verifica el código para éstos valores ($n = 3$ y $m = 10$).

Soluciones

Solución Ejercicio 1: Serie Condicional

```r
serie_condicional <- function(n) {
  suma <- 0
  for (i in 1:n) {
    if (i %% 2 == 0) {
      suma <- suma + 2 * i
    } else {
      suma <- suma - i
    }
  }
  return(suma)
}
n <- 10
resultado_serie <- serie_condicional(n)
cat("El resultado de la serie condicional es:", resultado_serie, "\n")
```

Solución Ejercicio 2: Producto Acumulado hasta un Umbral

```r
producto_acumulado <- function(n, T) {
  producto <- 1
  i <- 1
  while (i <= n && producto <= T) {
    producto <- producto * i
    i <- i + 1
  }
  return(producto)
}
n <- 10
T <- 100
resultado_producto <- producto_acumulado(n, T)
cat("El producto acumulado hasta superar T es:", resultado_producto, "\n")
```

Solución Ejercicio 3: Evaluación de Función Polinómica

```r
evaluar_polinomio <- function(a, x) {
  n <- length(a)
  resultado <- 0
  for (i in 1:n) {
    resultado <- resultado + a[i] * (x ^ (i - 1))
  }
  return(resultado)
}
a <- c(1, -2, 3)  # P(x) = 1 - 2x + 3x^2
x <- 2
resultado_polinomio <- evaluar_polinomio(a, x)
cat("El valor del polinomio en x =", x, "es:", resultado_polinomio, "\n")
```

Solución Ejercicio 4: Aproximación de Raíz Cuadrada por Iteración

```r
aproximar_raiz <- function(N, epsilon = 0.0001) {
  x_ant <- N / 2
  x_act <- (x_ant + N / x_ant) / 2
  while (abs(x_act - x_ant) > epsilon) {
    x_ant <- x_act
    x_act <- (x_ant + N / x_ant) / 2
  }
  return(x_act)
}
N <- 25
resultado_raiz <- aproximar_raiz(N)
cat("La aproximación de la raíz cuadrada de", N, "es:", resultado_raiz, "\n")
```

Solución Ejercicio 5: Serie Fibonacci Condicional

```r
fibonacci_modificado <- function(n) {
  fib <- function(i) {
    if (i == 1) return(0)
    if (i == 2) return(1)
    if (i %% 2 == 0) {
      return(fib(i - 1) + fib(i - 2))
    } else {
      return(fib(i - 1) - fib(i - 2))
    }
  }
  resultado <- numeric(n)
  for (i in 1:n) {
    resultado[i] <- fib(i)
  }
  return(resultado)
}
n <- 10
resultado_fib <- fibonacci_modificado(n)
cat("Serie Fibonacci Modificada:", resultado_fib, "\n")
```

Solución Ejercicio 6: Estimación de e mediante Serie de Taylor

```r
estimar_ex <- function(x, n) {
  factorial <- function(k) {
    resultado <- 1
    for (i in 1:k) resultado <- resultado * i
    return(resultado)
  }
  ex <- 0
  for (k in 0:n) {
    ex <- ex + (x^k) / factorial(k)
  }
  return(ex)
}
x <- 1
n <- 10
resultado_ex <- estimar_ex(x, n)
cat("Aproximación de e^", x, ":", resultado_ex, "\n")
```

Solución Ejercicio 7: Integración Numérica por Sumas de Riemann

```r
integracion_riemann <- function(f, a, b, n) {
  dx <- (b - a) / n
  suma <- 0
  for (i in 1:n) {
    x <- a + (i - 0.5) * dx
    suma <- suma + f(x) * dx
  }
  return(suma)
}
f <- function(x) x^2
a <- 0
b <- 1
n <- 1000
resultado_integracion <- integracion_riemann(f, a, b, n)
cat("Integral aproximada:", resultado_integracion, "\n")
```

Solución Ejercicio 8: Método de Bisección para Raíces

```r
biseccion <- function(f, a, b, tol, max_iter) {
  #Comprobamos que f(a) y f(b) no comparten signo:
  #la funcion ha cruzado por 0 almenos una vez en elintervalo
  if (f(a) * f(b) >= 0) {
    cat("No se puede aplicar el método en el intervalo dado\n")
    return(NA)
  }

  c <- (a + b) / 2
  iter <- 0  # Contador de iteraciones

  while ((b - a) / 2 > tol && iter < max_iter) {
    if (f(c) == 0) {
      break
    }

    if (f(a) * f(c) < 0) {
      b <- c
    } else {
      a <- c
    }

    c <- (a + b) / 2
    iter <- iter + 1  # Aumentar el contador de iteraciones
  }

  # Comprobar si el número máximo de iteraciones fue alcanzado
  if (iter == max_iter) {
    cat("Se alcanzó el número máximo de iteraciones\n")
  }

  return(c)
}

# Ejemplo de uso
f <- function(x) x^3 - 4  # Función de ejemplo
a <- 1  # Límite inferior del intervalo
b <- 2  # Límite superior del intervalo
```

```r
tol <- 0.001  # Tolerancia
max_iter <- 100  # Máximo número de iteraciones

resultado_raiz <- biseccion(f, a, b, tol, max_iter)
cat("Raíz aproximada:", resultado_raiz, "\n")
```

Solución Ejercicio 9: Determinación de Números Primos

```r
es_primo <- function(n) {
  if (n <= 1) return(FALSE)
  for (i in 2:(n - 1)) {
    if (n %% i == 0) return(FALSE)
  }
  return(TRUE)
}

# Verificacion de la función
n <- 17
resultado_primo <- es_primo(n)
cat("El número", n, "es primo:", resultado_primo, "\n")
n <- 18
resultado_primo <- es_primo(n)
cat("El número", n, "es primo:", resultado_primo, "\n")
```

Solución Ejercicio 10: Cálculo del Máximo Común Divisor (MCD)

```r
mcd <- function(a, b) {
  while (b != 0) {
    temp <- b
    b <- a %% b
    a <- temp
  }
  return(a)
}
a <- 56
b <- 98
resultado_mcd <- mcd(a, b)
cat("El MCD de", a, "y", b, "es:", resultado_mcd, "\n")
```

Solución Ejercicio 11: Generación de una Tabla de Multiplicar

```r
tabla_multiplicar <- function(n, m) {
  for (i in 1:m) {
    cat(n, "x", i, "=", n * i, "\n")
  }
}
n <- 3
m <- 10
cat (" Tabla de multiplicar de" , n , ":","\n" )
tabla_multiplicar(n, m)
```