Ejercicio 1

Sistema con Observaciones Inciertas:

Consideramos el sistema con observaciones inciertas definido en el Tema 4:

$$\begin{aligned} x(k+1) &= \Phi(k+1,k)x(k) + \Gamma(k+1,k)w(k), \quad k \geq 0; x(0) = x_0 \\ z(k) &= \gamma(k)H(k)x(k) + v(k), \quad k \geq 0 \end{aligned}$$

donde se verifican las siguientes hipótesis:

1. *El estado inicial, x_0, es un vector aleatorio n-dimensional gaussiano con media cero y matriz de covarianzas $E[x_0 x_0^T] = P_0$.*
2. *El proceso $\{w(k); k \geq 0\}$ es una sucesión ruido blanco gaussiana, centrada, con matrices de covarianzas $E[w(k)w^T(k)] = Q(k)$, $k \geq 0$.*
3. *El ruido multiplicativo $\{\gamma(k); k \geq 0\}$ es una sucesión de variables aleatorias independientes de Bernoulli, con $P(\gamma(k) = 1) = p(k)$.*
4. *El proceso $\{v(k); k \geq 0\}$ es un ruido blanco gaussiano, centrado y con covarianzas $E[v(k)v^T(k)] = R(k)$, $k \geq 0$, siendo $R(k)$ una matriz definida positiva.*
5. *El estado inicial x_0 y los ruidos $\{w(k); k \geq 0\}$, $\{v(k); k \geq 0\}$, $\{\gamma(k); k \geq 0\}$ son mutuamente independientes.*

Apartado (a)

Demostraremos que la matriz de covarianzas $\Pi(k) = E[\tilde{z}(k/k-1)\tilde{z}^T(k/k-1)]$ del proceso de innovación verifica:

$$\begin{aligned} \Pi(k) &= p(k)\big(1-p(k)\big)H(k)D(k)H^T(k) + p^2(k)H(k)P(k/k-1)H^T(k) + R(k) \\ \Pi(0) &= p(0)H(0)P_0H^T(0) + R(0), \end{aligned}$$

donde $D(k) = E[x(k)x^T(k)]$ y $P(k/k-1) = E[\tilde{x}(k/k-1)\tilde{x}^T(k/k-1)]$ son dadas por:

$$\begin{aligned} D(k) &= \Phi(k,k-1)D(k-1)\Phi^T(k,k-1) + \Gamma(k,k-1)Q(k-1)\Gamma^T(k,k-1) \\ D(0) &= P_0 \\ P(k/k-1) &= \Phi(k,k-1)P(k-1/k-1)\Phi^T(k,k-1) + \Gamma(k,k-1)Q(k-1)\Gamma^T(k,k-1), \quad k > 0 \\ P(0/-1) &= P_0 \\ P(k/k) &= E[\tilde{x}(k/k)\tilde{x}^T(k/k)]. \end{aligned}$$

Solución:

Vamos a demostrar tanto para $\Pi(k)$ como para $\Pi(0)$

En el caso individual, si $k = 0$: [Este resultado se debe a la independencia de los tres ruidos]

$$\Pi(0) = E[\tilde{z}(0/-1)\tilde{z}^T(0/-1)] = p(0)H(0)P_0H^T(0) + R(0)$$

Para el caso general ($k > 0$). Por el Lema de Proyecciones Ortogonales sabemos que

$$\hat{z}(k/k-1)$$

es el único elemento del subespacio generado por las observaciones

$$z(0), \ldots, z(k-1)$$

que verifica:

$$E[z(k)z^T(\alpha)] = E[\hat{z}(k/k-1)z^T(\alpha)$$

Por la ecuación de la observación y teniendo en cuenta que la sucesión de variables aleatorias independientes es a su vez independiente, se tiene que:

$$E[z(k)z^T(\alpha)] = p(k)H(k)E[x(k)z^T(\alpha) + E[v(k)z^T(\alpha)$$

Considerando el Lema de Proyecciones Ortogonales se puede llegar a:

$$\hat{z}(k/k-1) = p(k)H(k)\hat{x}(k/k-1) + \hat{v}(k/k-1)$$

Por lo tanto, el proceso innovador está dado por:

$$\delta(k) = z(k) - p(k)H(k)\hat{x}(k/k-1) - \hat{v}(k/k-1)$$

Si estudiamos las matrices de covarianzas tendremos $\Pi(k) = E[\delta(k)\delta^T(k)]$. Dado que $\hat{z}(k/k-1)$ es ortogonal a $\delta(k)$, tenemos:

$$\Pi(k) = E[z(k)\delta^T(k)]$$

Sustituyendo la expresión recién calculada:

$$\Pi(k) = E[z(k)z^T(k)] - p(k)E[z(k)\hat{x}^T(k/k-1)]H^T(k) - E[z(k)\hat{v}^T(k/k-1)]$$

Y por las propiedades del modelo citadas en el enunciado:

$$E[z(k)z^T(k)]$$
$$= p(k)H(k)D(k)H^T(k) + p(k)E[v(k)x^T(k)]H^T(k) + p(k)H(k)E[x(k)v^T(k)] + E[v(k)v^T(k)]$$

Y posteriormente:

$$E[z(k)\hat{x}^T(k/k-1)] = p(k)H(k)E[x(k)\hat{x}^T(k/k-1)] + E[v(k)\hat{x}^T(k/k-1)]$$

$$E[z(k)\hat{v}^T(k/k-1)] = p(k)H(k)E[x(k)\hat{v}^T(k/k-1)] + E[v(k)\hat{v}^T(k/k-1)]$$

Sustituyendo estas expresiones, agrupando términos y teniendo en cuenta que los estimadores $\hat{x}(k/k-1), \hat{v}(k/k-1)$

son ortogonales a los errores $e_x(k/k-1), e_v(k/k-1)$ se obtiene la expresión buscada de $\Pi(k)$

$$\Pi(k) = p(k)\big(1-p(k)\big)H(k)D(k)H^T(k) + p^2(k)H(k)P(k/k-1)H^T(k) + R(k)$$ ■

Apartado (b)

Se debe partir de la expresión del filtro de estado en función del predictor de una etapa:

$$\hat{x}(k/k) = \hat{x}(k/k-1) + F(k)[z(k) - \hat{z}(k/k-1)], k \geq 0$$

Si se sustituye aquí la expresión para la innovación se llega a:

$$\hat{x}(k/k) = \hat{x}(k/k-1) + F(k)[z(k) - p(k)H(k)\hat{x}(k/k-1) ~~- \hat{v}(k/k-1)~~]$$

Para obtener la matriz de ganancia, partimos de nuevo de la expresión de filtro de estado, ya que el error de filtrado verifica:

$$e_x(k/k) = e_x(k/k-1) - F(k)\delta(k)$$

Ya que $e_x(k/k)$ es ortogonal a la observación z(k):

$$E[e_x(k/k)z^T(k)] = 0$$

Y por lo tanto, se llega:

$$E[e_x(k/k-1)z^T(k)] = F(k)\Pi(k)$$

Se sustituye:

$$F(k) = E[e_x(k/k-1)z^T(k)]\Pi^{-1}(k)$$

Ahora, se puede calcular $E[e_x(k/k-1)z^T(k)]$ usando la ecuación de observación y las premisas iniciales:

$$E[e_x(k/k-1)z^T(k)] = p(k)E[e_x(k/k-1)z^T(k)]H^T(k) + E[e_x(k/k-1)z^T(k)]v^T(k)$$

Y como los errores de estimación son ortogonales a los estimadores (LPO):

$$E[e_x(k/k-1)z^T(k)] = p(k)P(k/k-1)H^T(k) + P_{xv}(k, k/k-1)$$

Si sustituimos en la ecuación calculada anteriormente, llegamos a la expresión buscada para F(k):

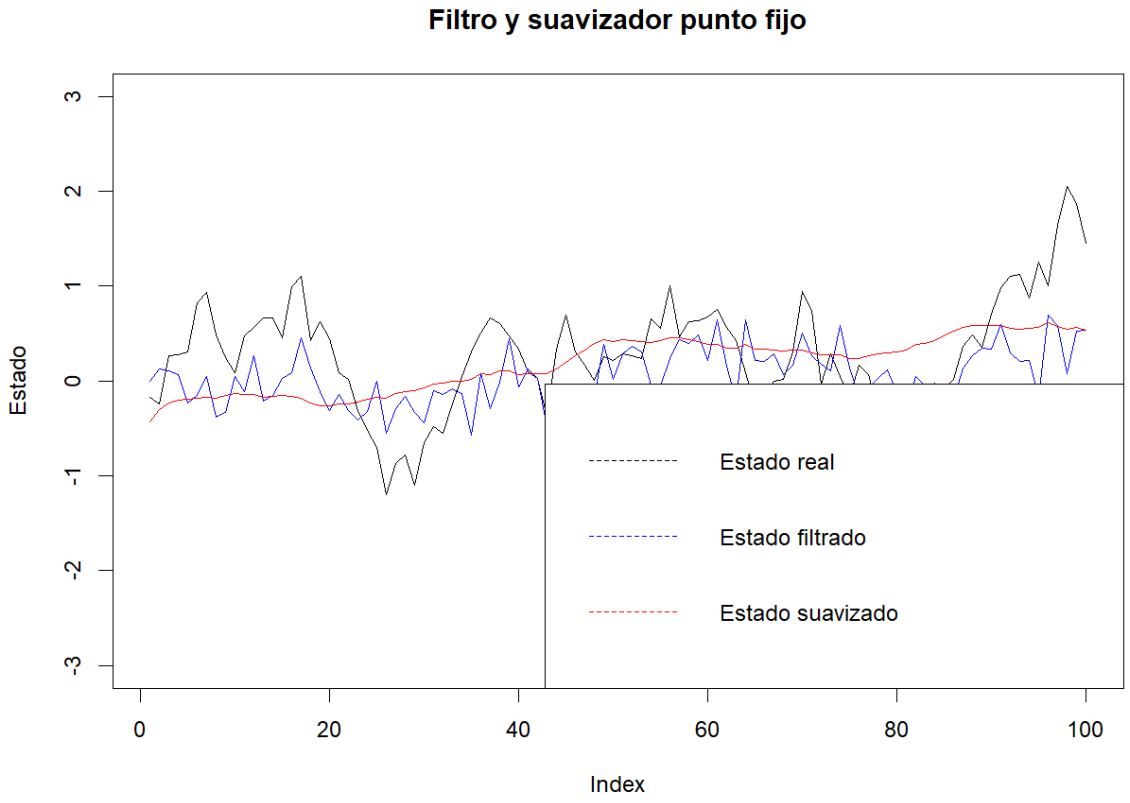
$$F(k) = p(k)P(k/k-1)H^T(k) + P_{xv}(k, k/k-1)\Pi^{-1}(k) \blacksquare$$

Apartado (c)

```
# Definir los parámetros del sistema
x0 <- 0 # Estado inicial
P0 <- 1 # Varianza inicial
Q <- 0.1 # Varianza del ruido de proceso
R <- 0.5 # Varianza del ruido de medida
p <- 0.5 # Probabilidad del ruido multiplicativo
phi <- 0.95 # Coeficiente del modelo del sistema

# Definir el número de pasos
```

```r
N <- 100

# Generar las medidas simuladas
set.seed(123) # Fijar la semilla para reproducibilidad
w <- rnorm(N, mean = 0, sd = sqrt(Q)) # Ruido de proceso
v <- rnorm(N, mean = 0, sd = sqrt(R)) # Ruido de medida
gamma <- rbinom(N, size = 1, prob = p) # Ruido multiplicativo
x <- numeric(N) # Estado real
z <- numeric(N) # Medida
x[1] <- x0 + w[1] # Inicializar el estado real
z[1] <- gamma[1] * x[1] + v[1] # Inicializar la medida
for (k in 2:N) {
  x[k] <- phi * x[k-1] + w[k] # Actualizar el estado real
  z[k] <- gamma[k] * x[k] + v[k] # Actualizar la medida
}

# Inicializar los vectores y matrices para el filtro y el suavizador
xf <- numeric(N) # Estado filtrado
xs <- numeric(N) # Estado suavizado
Pf <- numeric(N) # Varianza del estado filtrado
Ps <- numeric(N) # Varianza del estado suavizado
K <- numeric(N) # Ganancia de Kalman
J <- numeric(N) # Ganancia de suavización

# Aplicar el ciclo computacional del filtro y el suavizador
xf[1] <- x0 # Inicializar el estado filtrado
Pf[1] <- P0 # Inicializar la varianza del estado filtrado
for (k in 2:N) {
  # Predicción
  xf[k] <- phi * xf[k-1] # Predecir el estado
  Pf[k] <- phi^2 * Pf[k-1] + Q # Predecir la varianza
  # Actualización
  K[k] <- p * Pf[k] / (p^2 * Pf[k] + R) # Calcular la ganancia de Kalman
  xf[k] <- xf[k] + K[k] * (z[k] - xf[k]) # Actualizar el estado filtrado
  Pf[k] <- (1 - p * K[k]) * Pf[k] # Actualizar la varianza del estado
filtrado
}
xs[N] <- xf[N] # Inicializar el estado suavizado
Ps[N] <- Pf[N] # Inicializar la varianza del estado suavizado
for (k in (N-1):1) {
  # Suavización
  J[k] <- Pf[k] * phi / Pf[k+1] # Calcular la ganancia de suavización
  xs[k] <- xf[k] + J[k] * (xs[k+1] - phi * xf[k]) # Actualizar el estado
suavizado
  Ps[k] <- Pf[k] + J[k] * (Ps[k+1] - Pf[k+1]) * J[k] # Actualizar la
varianza del estado suavizado
}

# Mostrar los resultados en una gráfica
```

```
plot(x, type = "l", col = "black", ylim = c(-3, 3), ylab = "Estado", main
= "Filtro y suavizador punto fijo")
lines(xf, col = "blue")
lines(xs, col = "red")
legend("bottomright", legend = c("Estado real", "Estado filtrado",
"Estado suavizado"), col = c("black", "blue", "red"), lty = 1)
```



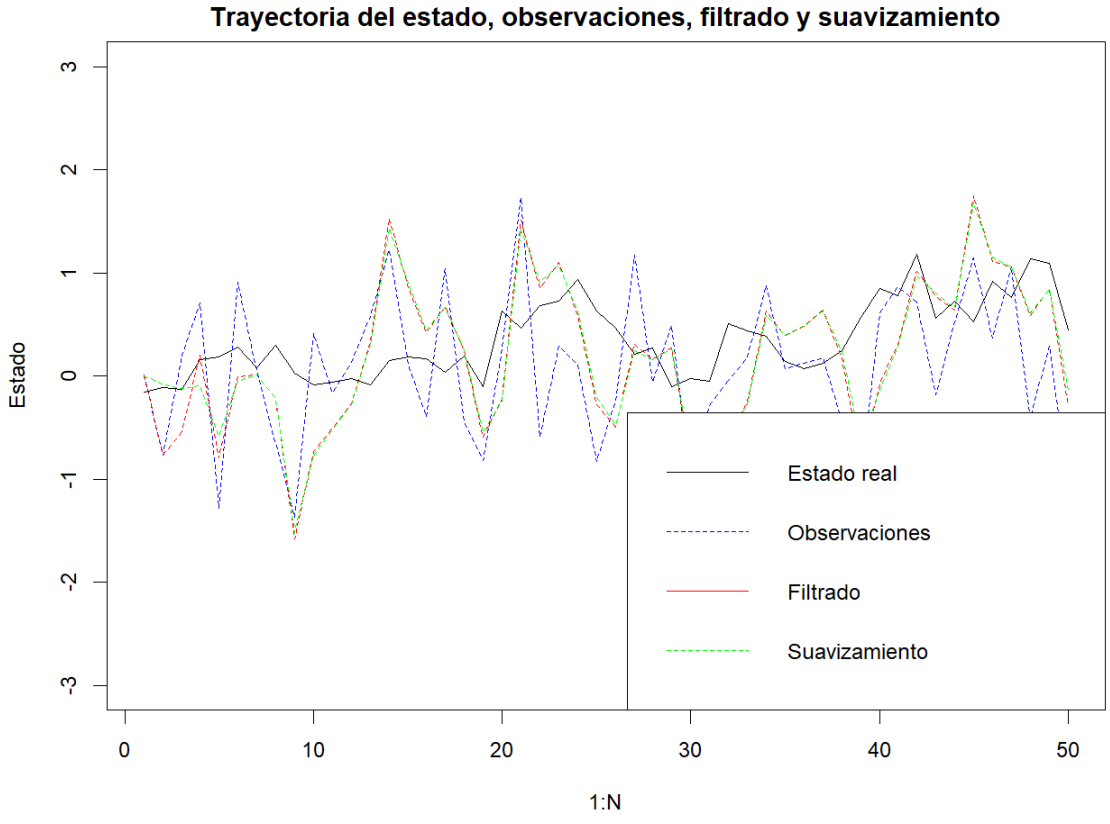
Apartado (d)

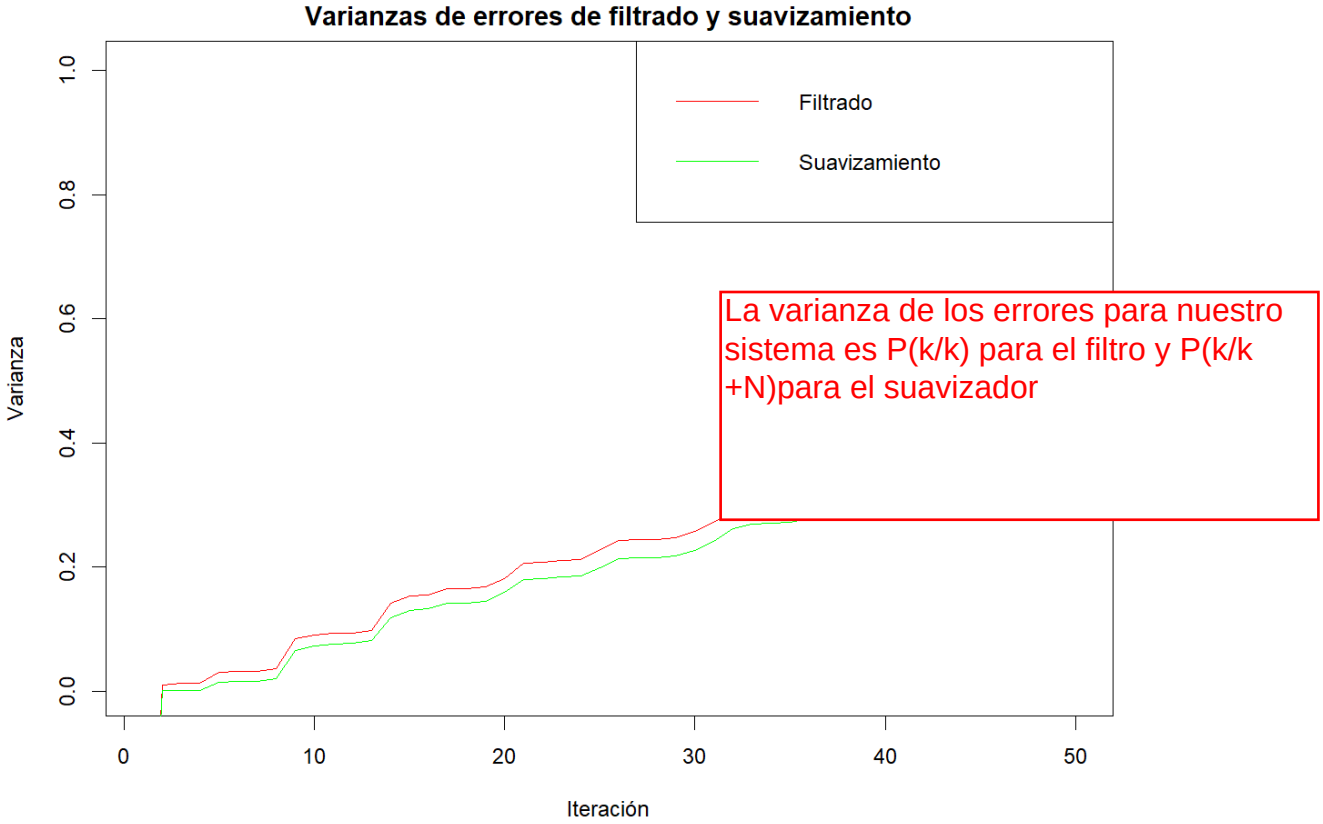
Repasar todas las expresiones del algoritmo de filtrado y suavizamiento. Comprueba que coinciden con las dadas en el Tema 4 para el filtro y con las he puesto en la plataforma para el suavizador.

```
# Definir parámetros del sistema
n <- 1       # Dimensión del estado
N <- 50      # Número de iteraciones
x0 <- 0      # Media del estado inicial
P0 <- 1      # Varianza del estado inicial
phi <- 0.95  # Coeficiente del modelo del sistema
Q <- 0.1     # Varianza del ruido de proceso
p <- 0.5     # Probabilidad del ruido multiplicativo
R <- 0.5     # Varianza del ruido de medida

# Inicializar vectores y matrices
x <- numeric(N)  # Estado real
z <- numeric(N)  # Observaciones
x_filt <- numeric(N)  # Estimaciones de filtrado
x_smooth <- numeric(N)  # Estimaciones de suavizamiento
```

```
P_filt <- numeric(N)  # Varianzas de filtrado
P_smooth <- numeric(N)  # Varianzas de suavizamiento
K <- numeric(N)  # Ganancia de Kalman
J <- numeric(N)  # Ganancia de suavizamiento

# Generar observaciones inciertas
set.seed(100)
w <- rnorm(N, mean = 0, sd = sqrt(Q))
gamma <- rbinom(N, size = 1, prob = p)
v <- rnorm(N, mean = 0, sd = sqrt(R))
x[1] <- x0 + w[1]
z[1] <- gamma[1] * x[1] + v[1]
x_filt[1] <- 0 # Estimación inicial
P_filt[1] <- 1

# Aplicar el ciclo computacional del filtro y el suavizador
for (k in 2:N) {
  # Modelo del sistema
  x[k] <- phi * x[k-1] + w[k]
  z[k] <- gamma[k] * x[k] + v[k]

  # Filtro de Kalman
  # Predicción
  x_filt[k] <- phi * x_filt[k-1]
  P_filt[k] <- phi^2 * P_filt[k-1] + Q

  # Actualización
  K[k] <- P_filt[k] / (P_filt[k] + gamma[k]^2 * R)
  x_filt[k] <- x_filt[k] + K[k] * (z[k] - gamma[k] * x_filt[k])
  P_filt[k] <- (1 - K[k] * gamma[k]) * P_filt[k]

  # Suavizador
  if (k >= 2) {
    J[k-1] <- P_filt[k-1] * phi * P_filt[k]
    x_smooth[k] <- x_filt[k] + J[k-1] * (x_smooth[k-1] - phi * x_filt[k])
    P_smooth[k] <- P_filt[k] + J[k-1] * (P_smooth[k-1] - P_filt[k]) *
J[k-1]

    # Cálculo de errores
    e_filt <- x_filt - x
    e_suav <- x_smooth - x

    # Cálculo de varianzas
    e_filt2[k] <- var(e_filt)
    e_suav2[k] <- var(e_suav)
  }
}

# Gráfica 1: Trayectoria del estado, observaciones, estimaciones de
```

```
filtrado y suavizamiento
par(mfrow = c(1, 1))
par(mar = c(4, 4, 2, 1))  # Ajustar los márgenes

plot(1:N, x, type = "l", col = "black", ylim = c(-3, 3), ylab = "Estado",
     main = "Trayectoria del estado, observaciones, filtrado y
suavizamiento")
lines(1:N, z, col = "blue", lty = 2)
lines(1:N, x_filt, col = "red", lty = 2)
lines(1:N, x_smooth, col = "green", lty = 2)
legend("bottomright", legend = c("Estado real", "Observaciones",
"Filtrado", "Suavizamiento"),
       col = c("black", "blue", "red", "green"), lty = 1:2)

# Gráfica 2: Varianzas de los errores de filtrado y suavizamiento
par(mar = c(4, 4, 2, 1))  # Ajustar los márgenes

plot(1:N, e_filt2, type = "l", col = "red", ylim = c(0, max(P_filt,
P_smooth)), ylab = "Varianza",
     xlab = "Iteración", main = "Varianzas de errores de filtrado y
suavizamiento")
lines(1:N, e_suav2, col = "green")
legend("topright", legend = c("Filtrado", "Suavizamiento"), col =
c("red", "green"), lty = 1)
```





Para representar las varianzas de errores con N=1, N=2 y N=4 se puede adaptar el script anterior:

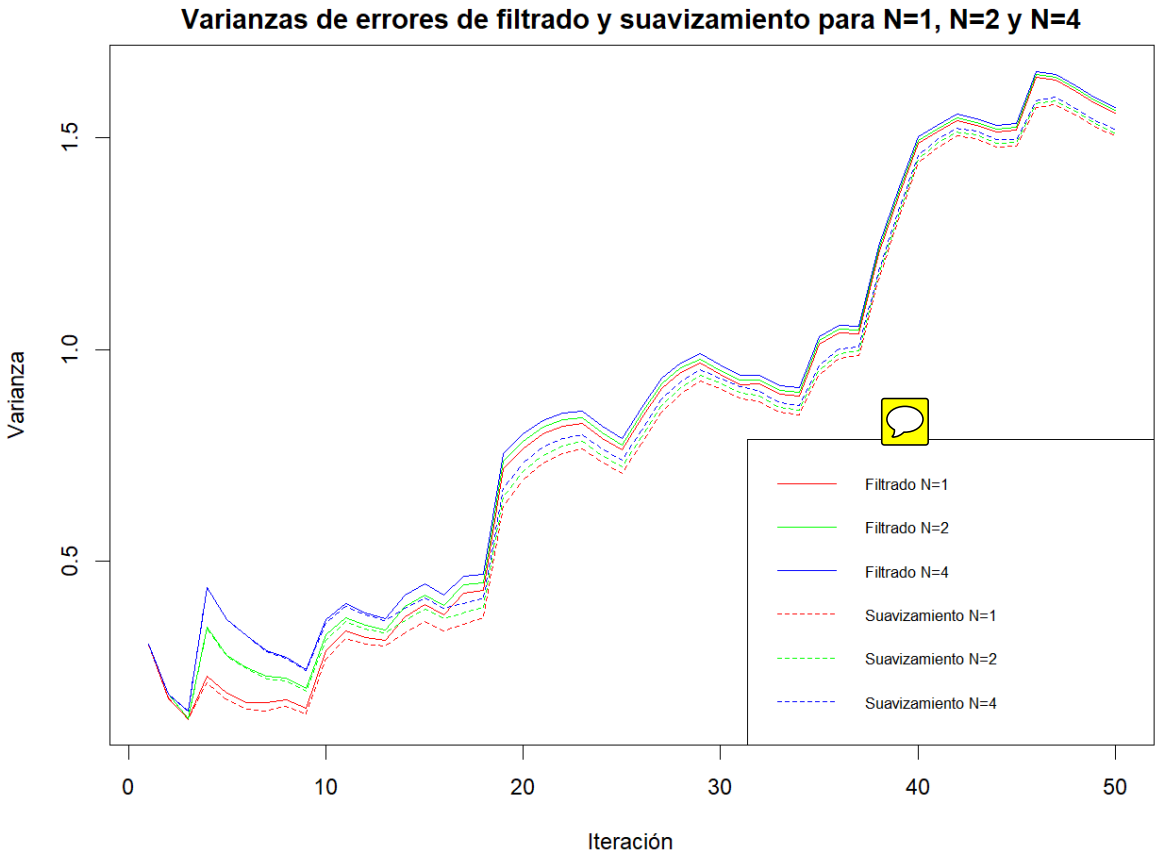
```
# Definir parámetros del sistema
n <- 1       # Dimensión del estado
N <- 50      # Número de iteraciones
x0 <- 0      # Media del estado inicial
P0 <- 1      # Varianza del estado inicial
phi <- 0.95  # Coeficiente del modelo del sistema
Q <- 0.1     # Varianza del ruido de proceso
p <- 0.5     # Probabilidad del ruido multiplicativo
R <- 0.5     # Varianza del ruido de medida

# Inicializar vectores y matrices
x <- numeric(N)  # Estado real
z <- numeric(N)  # Observaciones
x_filt1 <- numeric(N)  # Estimaciones de filtrado para N=1
x_filt2 <- numeric(N)  # Estimaciones de filtrado para N=2
x_filt4 <- numeric(N)  # Estimaciones de filtrado para N=4
x_smooth1 <- numeric(N)  # Estimaciones de suavizamiento para N=1
x_smooth2 <- numeric(N)  # Estimaciones de suavizamiento para N=2
x_smooth4 <- numeric(N)  # Estimaciones de suavizamiento para N=4
P_filt1 <- numeric(N)  # Varianzas de filtrado para N=1
P_filt2 <- numeric(N)  # Varianzas de filtrado para N=2
P_filt4 <- numeric(N)  # Varianzas de filtrado para N=4
```

```
P_smooth1 <- numeric(N)  # Varianzas de suavizamiento para N=1
P_smooth2 <- numeric(N)  # Varianzas de suavizamiento para N=2
P_smooth4 <- numeric(N)  # Varianzas de suavizamiento para N=4
K1 <- numeric(N)  # Ganancia de Kalman para N=1
K2 <- numeric(N)  # Ganancia de Kalman para N=2
K4 <- numeric(N)  # Ganancia de Kalman para N=4
J1 <- numeric(N)  # Ganancia de suavizamiento para N=1
J2 <- numeric(N)  # Ganancia de suavizamiento para N=2
J4 <- numeric(N)  # Ganancia de suavizamiento para N=4

# Generar observaciones inciertas
set.seed(50)
w <- rnorm(N, mean = 0, sd = sqrt(Q))
gamma <- rbinom(N, size = 1, prob = p)
v <- rnorm(N, mean = 0, sd = sqrt(R))
x[1] <- rnorm(1)
z[1] <- x[1] + sqrt(R) * rnorm(1)
x_filt1[1] <- 0  # Estimación inicial para N=1
x_filt2[1] <- 0  # Estimación inicial para N=2
x_filt4[1] <- 0  # Estimación inicial para N=4
P_filt1[1] <- 1
P_filt2[1] <- 1
P_filt4[1] <- 1
x_smooth1[1] <- 0  # Estimación inicial para N=1
x_smooth2[1] <- 0  # Estimación inicial para N=2
x_smooth4[1] <- 0  # Estimación inicial para N=4
P_smooth1[1] <- 1
P_smooth2[1] <- 1
P_smooth4[1] <- 1

# Aplicar el ciclo computacional del filtro y el suavizador para N=1
for (k in 2:N) {
  # Modelo del sistema
  x[k] <- phi * x[k-1] + w[k]
  z[k] <- gamma[k] * x[k] + v[k]

  # Filtro de Kalman
  # Predicción
  x_filt1[k] <- phi * x_filt1[k-1]
  P_filt1[k] <- phi^2 * P_filt1[k-1] + Q

  # Actualización
  K1[k] <- P_filt1[k] / (P_filt1[k] + gamma[k]^2 * R)
  x_filt1[k] <- x_filt1[k] + K1[k] * (z[k] - gamma[k] * x_filt1[k])
  P_filt1[k] <- (1 - K1[k] * gamma[k]) * P_filt1[k]

  # Suavizador
  if (k >= 2) {
    J1[k-1] <- P_filt1[k-1] * phi * P_filt1[k]
```

```
    x_smooth1[k] <- x_filt1[k] + J1[k-1] * (x_smooth1[k-1] - phi *
x_filt1[k])
    P_smooth1[k] <- P_filt1[k] + J1[k-1] * (P_smooth1[k-1] - P_filt1[k])
* J1[k-1]
  }
}

# Aplicar el ciclo computacional del filtro y el suavizador para N=2
for (k in 3:N) {
  # Modelo del sistema
  x[k] <- phi * x[k-1] + w[k]
  z[k] <- gamma[k] * x[k] + v[k]

  # Filtro de Kalman
  # Predicción
  x_filt2[k] <- phi * x_filt2[k-1]
  P_filt2[k] <- phi^2 * P_filt2[k-1] + Q

  # Actualización
  K2[k] <- P_filt2[k] / (P_filt2[k] + gamma[k]^2 * R)
  x_filt2[k] <- x_filt2[k] + K2[k] * (z[k] - gamma[k] * x_filt2[k])
  P_filt2[k] <- (1 - K2[k] * gamma[k]) * P_filt2[k]

  # Suavizador
  if (k >= 3) {
    J2[k-1] <- P_filt2[k-1] * phi * P_filt2[k]
    x_smooth2[k] <- x_filt2[k] + J2[k-1] * (x_smooth2[k-1] - phi *
x_filt2[k])
    P_smooth2[k] <- P_filt2[k] + J2[k-1] * (P_smooth2[k-1] - P_filt2[k])
* J2[k-1]
  }
}

# Aplicar el ciclo computacional del filtro y el suavizador para N=4
for (k in 4:N) {
  # Modelo del sistema
  x[k] <- phi * x[k-1] + w[k]
  z[k] <- gamma[k] * x[k] + v[k]

  # Filtro de Kalman
  # Predicción
  x_filt4[k] <- phi * x_filt4[k-1]
  P_filt4[k] <- phi^2 * P_filt4[k-1] + Q

  # Actualización
  K4[k] <- P_filt4[k] / (P_filt4[k] + gamma[k]^2 * R)
  x_filt4[k] <- x_filt4[k] + K4[k] * (z[k] - gamma[k] * x_filt4[k])
  P_filt4[k] <- (1 - K4[k] * gamma[k]) * P_filt4[k]
```

```
  # Suavizador
  if (k >= 4) {
    J4[k-1] <- P_filt4[k-1] * phi * P_filt4[k]
    x_smooth4[k] <- x_filt4[k] + J4[k-1] * (x_smooth4[k-1] - phi *
x_filt4[k])
    P_smooth4[k] <- P_filt4[k] + J4[k-1] * (P_smooth4[k-1] - P_filt4[k])
* J4[k-1]
  }
}

# Cálculo de errores
e_filt1 <- x_filt1 - x
e_filt2 <- x_filt2 - x
e_filt4 <- x_filt4 - x
e_suav1 <- x_smooth1 - x
e_suav2 <- x_smooth2 - x
e_suav4 <- x_smooth4 - x

# Cálculo de varianzas
e_filt1_var <- cumsum(e_filt1^2) / (1:N)
e_suav1_var <- cumsum(e_suav1^2) / (1:N)
e_filt2_var <- cumsum(e_filt2^2) / (1:N)
e_suav2_var <- cumsum(e_suav2^2) / (1:N)
e_filt4_var <- cumsum(e_filt4^2) / (1:N)
e_suav4_var <- cumsum(e_suav4^2) / (1:N)

# Gráfica: Varianzas de los errores de filtrado y suavizamiento para N=1,
N=2 y N=4
par(mfrow = c(1, 1))
par(mar = c(4, 4, 2, 1))  # Ajustar los márgenes

# Calcular los límites del eje y
y_limits <- c(min(e_filt1_var, e_suav1_var, e_filt2_var, e_suav2_var,
e_filt4_var, e_suav4_var), max(e_filt1_var, e_suav1_var, e_filt2_var,
e_suav2_var, e_filt4_var, e_suav4_var))

plot(1:N, e_filt1_var, type = "l", col = "red", lty = 1, ylab =
"Varianza",
     xlab = "Iteración", main = "Varianzas de errores de filtrado y
suavizamiento para N=1, N=2 y N=4", ylim = y_limits)
lines(1:N, e_filt2_var, col = "green", lty = 1)
lines(1:N, e_filt4_var, col = "blue", lty = 1)
lines(1:N, e_suav1_var, col = "red", lty = 2)
lines(1:N, e_suav2_var, col = "green", lty = 2)
lines(1:N, e_suav4_var, col = "blue", lty = 2)
legend("bottomright", legend = c("Filtrado N=1", "Filtrado N=2",
"Filtrado N=4", "Suavizamiento N=1", "Suavizamiento N=2", "Suavizamiento
N=4"),
```

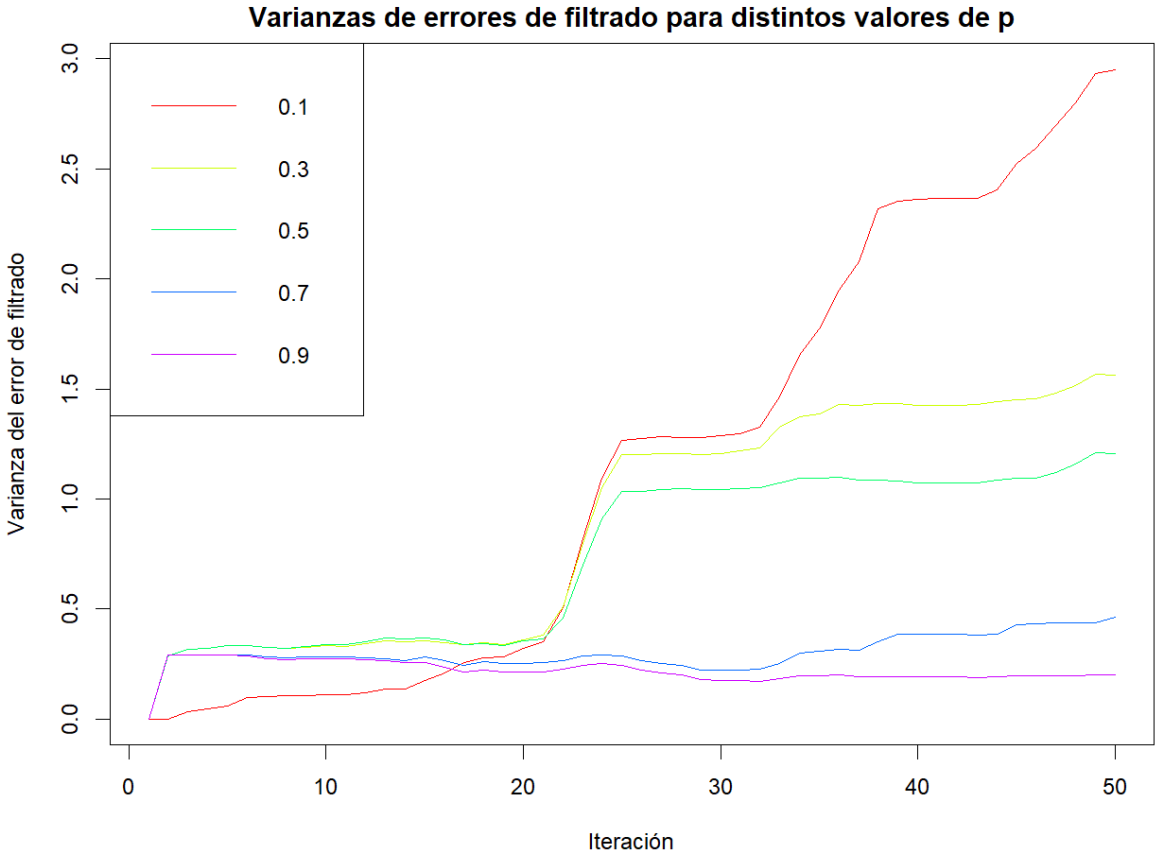
```
       col = c("red", "green", "blue", "red", "green", "blue"), lty =
c(1, 1, 1, 2, 2, 2), cex=0.7)
```



Apartado (e)

```
# Definir parámetros del sistema
n <- 1       # Dimensión del estado
N <- 50      # Número de iteraciones
x0 <- 0      # Media del estado inicial
P0 <- 1      # Varianza del estado inicial
phi <- 0.95  # Coeficiente del modelo del sistema
Q <- 0.1     # Varianza del ruido de proceso
p_values <- c(0.1, 0.3, 0.5, 0.7, 0.9)  # Valores de probabilidad del
ruido multiplicativo
R <- 0.5     # Varianza del ruido de medida

# Inicializar vectores y matrices
x <- numeric(N)  # Estado real
z <- numeric(N)  # Observaciones
e_filt2_values <- matrix(0, nrow = N, ncol = length(p_values))  #
Varianzas de errores de filtrado

# Aplicar el ciclo computacional del filtro para diferentes valores de p
for (p_index in seq_along(p_values)) {
```

```
  p <- p_values[p_index]

  # Inicializar vectores y matrices
  x_filt <- numeric(N)  # Estimaciones de filtrado
  P_filt <- numeric(N)  # Varianzas de filtrado
  K <- numeric(N)  # Ganancia de Kalman

  # Generar observaciones inciertas
  set.seed(123)
  w <- rnorm(N, mean = 0, sd = sqrt(Q))
  gamma <- rbinom(N, size = 1, prob = p)
  v <- rnorm(N, mean = 0, sd = sqrt(R))
  x[1] <- x0 + w[1]
  z[1] <- gamma[1] * x[1] + v[1]

  # Aplicar el ciclo computacional del filtro
  for (k in 2:N) {
    # Modelo del sistema
    x[k] <- phi * x[k-1] + w[k]
    z[k] <- gamma[k] * x[k] + v[k]

    # Filtro de Kalman
    # Predicción
    x_filt[k] <- phi * x_filt[k-1]
    P_filt[k] <- phi^2 * P_filt[k-1] + Q
    # Actualización
    K[k] <- P_filt[k] / (P_filt[k] + gamma[k]^2 * R)
    x_filt[k] <- x_filt[k] + K[k] * (z[k] - gamma[k] * x_filt[k])
    P_filt[k] <- (1 - K[k] * gamma[k]) * P_filt[k]

    # Cálculo de los errores de filtrado y su varianza
    e_filt <- x_filt - x
    e_filt2_values[k, p_index] <- var(e_filt)
  }
}

# Graficar las varianzas de los errores de filtrado para diferentes
valores de p
plot(1:N, e_filt2_values[, 1], type = "l", col = "red", ylim = c(0,
max(e_filt2_values)),
     ylab = "Varianza del error de filtrado", xlab = "Iteración", main =
"Varianzas de errores de filtrado para distintos valores de p")
for (i in 2:length(p_values)) {
  lines(1:N, e_filt2_values[, i], col = rainbow(length(p_values))[i])
}
legend("topleft", legend = c(p_values), col = rainbow(length(p_values)),
lty = 1)
```



Se observa una mayor varianza en los errores de filtrado para p=0.1 y p=0.3. Esto se debe a que el ruido multiplicativo tiene una menor probabilidad de ser activado, lo que significa que las observaciones están más influenciadas por el ruido de proceso.

Al mismo tiempo, se observa menor varianza para p=0.9. Las observaciones están más influenciadas por el ruido de medida, lo que reduce la incertidumbre en las estimaciones del filtro. Realmente, todas las varianzas se ordenan por orden teniendo en cuenta los valores de p.

Ejercicio 2

Sea {x(k); k ≥ 0} un proceso estocástico escalar definido mediante la relación

$$x(k+1) = (-1)^{2k+1}x(k), \quad k \geq 0$$

donde x_0 es una variable gaussiana con media 0 y varianza 1. Supongamos que disponemos de observaciones de este proceso de la forma

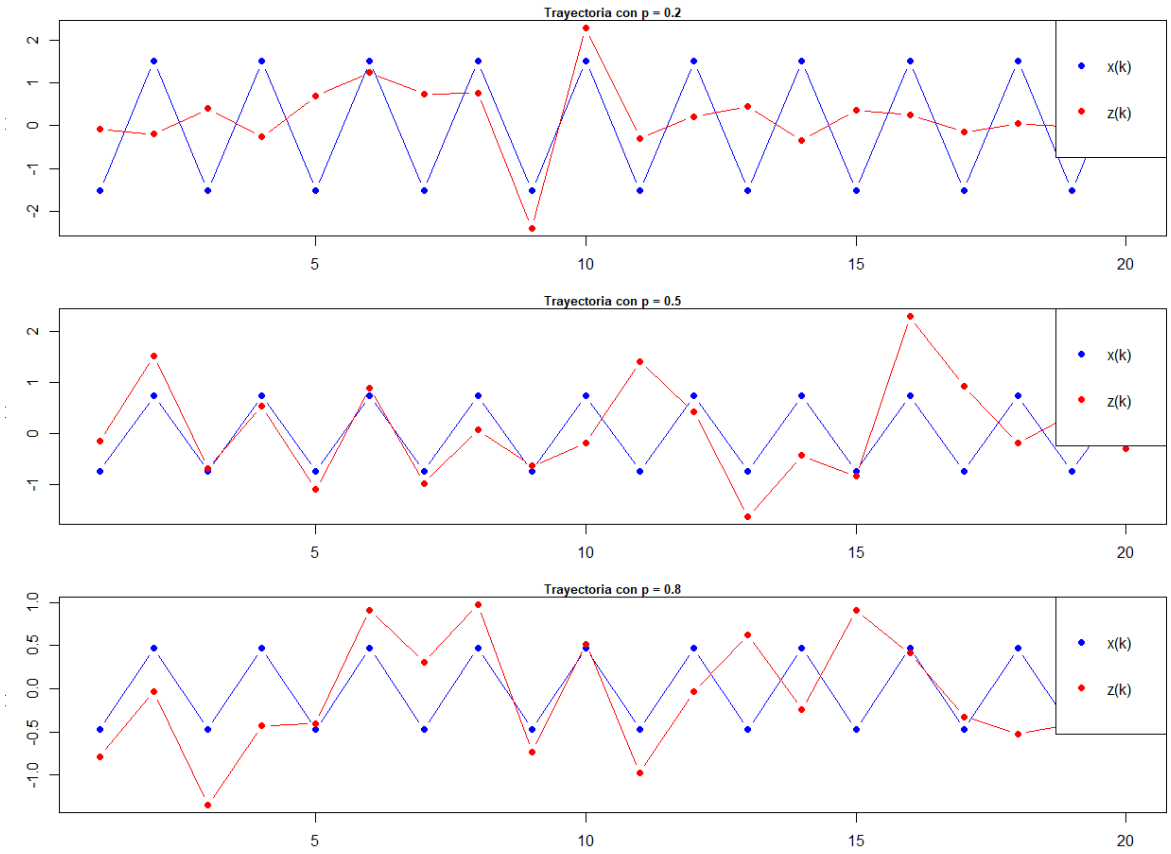
$$z(k) = \gamma(k)x(k) + v(k), \quad k \geq 0$$

donde el ruido multiplicativo $\{\gamma(k); k \geq 0\}$ es una sucesión de variables aleatorias independientes de Bernoulli, con $P(\gamma(k) = 1) = p$, y el ruido aditivo $\{v(k); k \geq 0\}$ es una sucesión blanca gaussiana, centrada y con varianzas $E[v^2(k)] = 0.5, \quad k \geq 0$.

Apartado (a)

(a) Representar los 20 primeros valores de dos trayectorias del proceso $\{x(k); k \geq 0\}$ y sus correspondientes valores observados, considerando distintos valores de la probabilidad p; comentar los resultados.

```
# Función para generar trayectorias del proceso
generar_trayectoria <- function(p, n = 20) {
  x <- numeric(n)
  x[1] <- rnorm(1)

  for (k in 2:n) {
    x[k] <- (-1)^(2*k + 1) * x[k - 1]
  }

  gamma <- rbinom(n, size = 1, prob = p)
  v <- rnorm(n, mean = 0, sd = sqrt(0.5))
  z <- gamma * x + v

  return(list(x = x, z = z))
}

# Configuración de la gráfica
par(mfrow = c(3, 1), mar = c(3, 3, 1, 1))  # Ajustar márgenes

# Valores de p a considerar
p_values <- c(0.2, 0.5, 0.8)

# Generar y graficar trayectorias para distintos valores de p
for (p in p_values) {
  trayectoria <- generar_trayectoria(p)

  # Gráfica de la trayectoria
  plot(1:20, trayectoria$x, type = 'b', pch = 19, col = 'blue', ylim =
range(trayectoria$x, trayectoria$z),
       main = sprintf('Trayectoria con p = %.1f', p), xlab = 'k', ylab =
'x(k)', cex.main = 0.8)

  # Gráfica de los valores observados
  points(1:20, trayectoria$z, type = 'b', pch = 19, col = 'red')

  # Leyenda
  legend('topright', legend = c('x(k)', 'z(k)'), col = c('blue', 'red'),
pch = 19)
}

# Restaurar configuración original de la gráfica
par(mfrow = c(1, 1))
```



x(k) es la variable real y z(k) es la observación correspondiente a esa variable real, afectada por un ruido y multiplicada por una variable aleatoria de Bernoulli.

Las observaciones del proceso son ruidosas y dependen del valor del ruido multiplicativo, que es una variable de Bernoulli. Cuando el ruido multiplicativo es cero, la observación es igual al ruido aditivo, que es una variable gaussiana centrada. Cuando el ruido multiplicativo es uno, la observación es igual al valor del proceso más el ruido aditivo. Por tanto, las observaciones pueden estar más o menos cerca de las trayectorias, dependiendo del valor de p, que es la probabilidad de que el ruido multiplicativo sea uno. Si p es cercano a cero, las observaciones estarán más dispersas y alejadas de las trayectorias. Si p es cercano a uno, las observaciones estarán más concentradas y cercanas a las trayectorias.

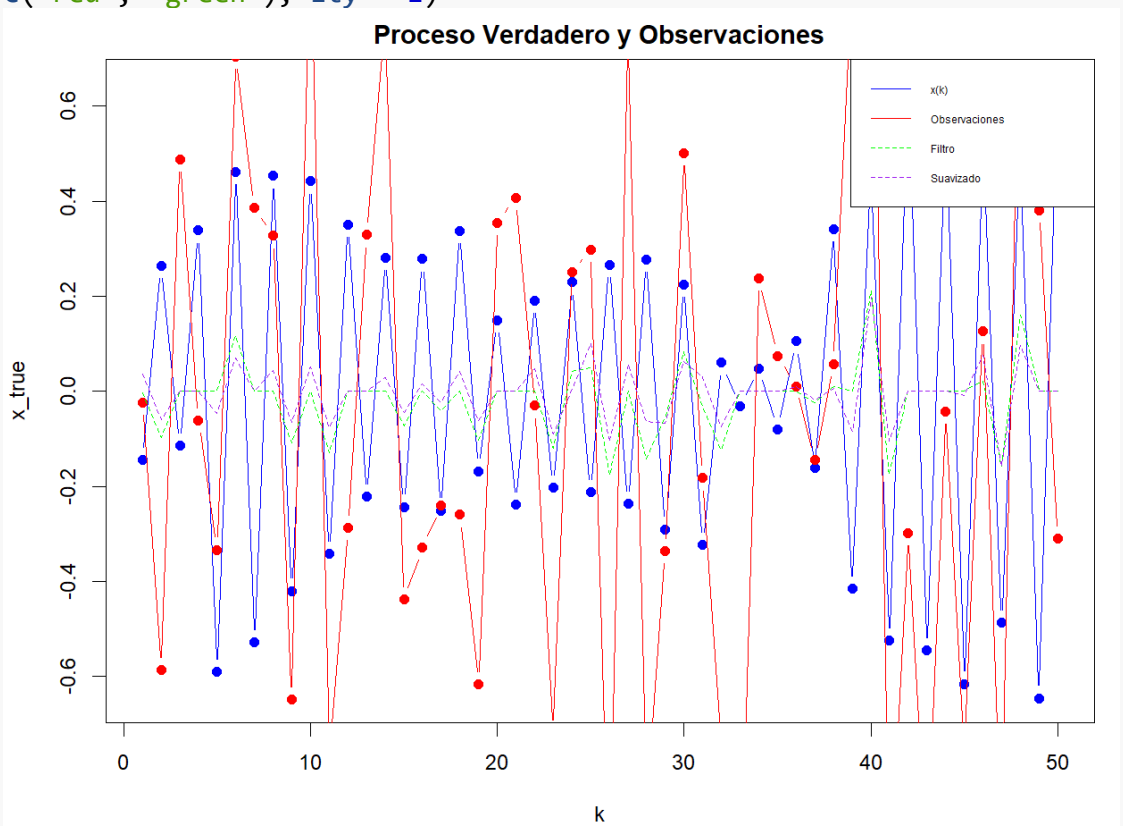
Apartado (b)

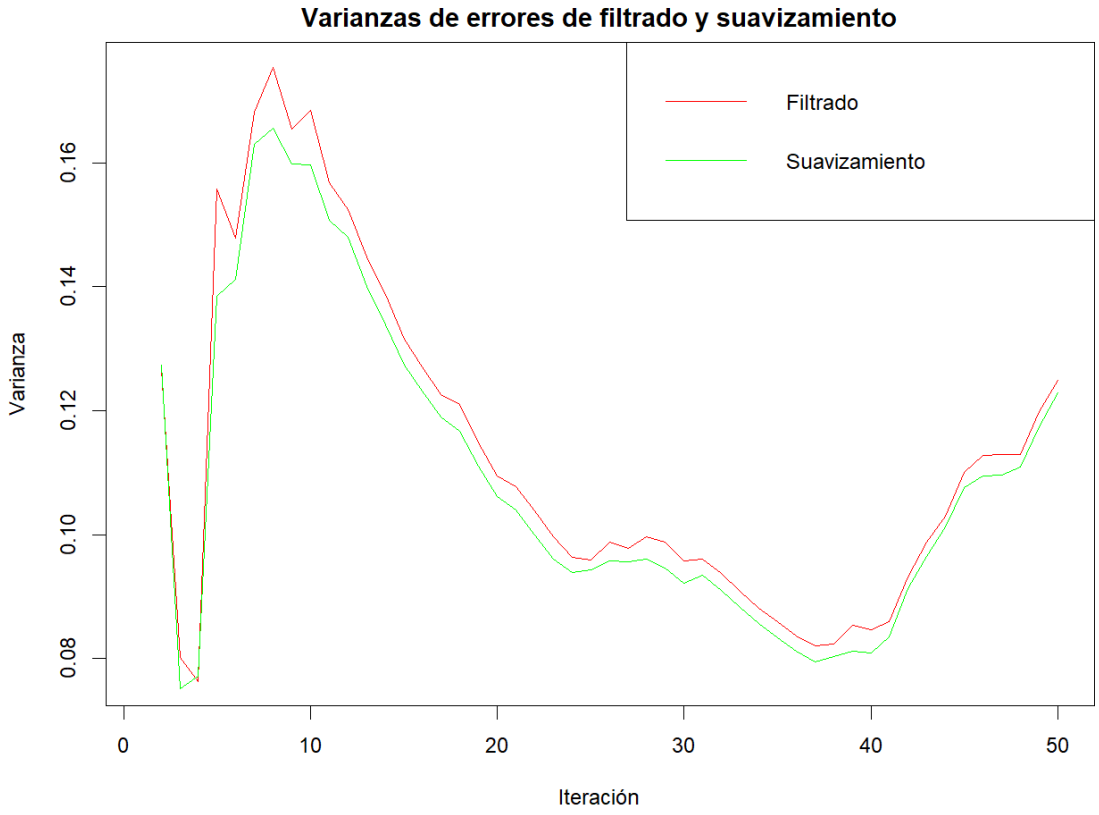
```
# Función de Filtrado de Kalman
filtro_kalman <- function(z, gamma, p, Q, R) {
  K <- length(z)

  # Inicialización
  x_filt <- rep(0, K)
  P_filt <- rep(0, K)

  # Algoritmo de Filtrado de Kalman
```

```
  for (k in 1:K) {
    # Predicción
    x_pred = (-1)^(2*k + 1) * x_filt[k]
    P_pred = abs((-1)^(2*k + 1)) * P_filt[k] + Q

    # Ganancia de Kalman
    K_gain = P_pred * gamma[k] / (gamma[k]^2 * P_pred + R)

    # Actualización
    x_filt[k] = x_pred + K_gain * (z[k] - gamma[k] * x_pred)
    P_filt[k] = (1 - K_gain * gamma[k]) * P_pred
  }

  return(list(x_filt = x_filt, P_filt = P_filt))
}
# Función de Suavizamiento Punto Fijo
suavizador_punto_fijo <- function(x_filt, P_filt, gamma, p, Q, R, N) {
  K <- length(x_filt)

  # Inicialización
  x_smooth <- x_filt
  P_smooth <- P_filt

  # Algoritmo de Suavizamiento Punto Fijo
  for (n in 1:N) {
    for (k in K:2) {
      # Suavizamiento
      A = P_filt[k - 1] / P_filt[k]
      x_smooth[k - 1] = 0.8 * x_filt[k - 1] - 0.2 * x_filt[k]
      P_smooth[k - 1] = P_filt[k - 1] + A^2 * (P_smooth[k - 1] - P_filt[k
- 1])
    }
  }

  return(list(x_smooth = x_smooth, P_smooth = P_smooth))
}

# Parámetros del sistema
K <- 50
x0 <- rnorm(1, mean = 0, sd = 1)
gamma <- rbinom(K, 1, 0.5)
w <- rnorm(K, mean = 0, sd = 0.1)
v <- rnorm(K, mean = 0, sd = 0.5)
p <- 0.5
Q <- 0.1
R <- 0.5

# Proceso verdadero
x_true <- rep(0, K)
```

```
x_true[1] <- x0
for (k in 2:K) {
  x_true[k] <- (-1)^(2*k + 1) * x_true[k - 1] + w[k - 1]
}

# Observaciones
z <- gamma * x_true + v

# Aplicar el algoritmo de Filtrado de Kalman
filtro_result <- filtro_kalman(z, gamma, p, Q, R)
x_filt <- filtro_result$x_filt
P_filt <- filtro_result$P_filt

# Aplicar el algoritmo de Suavizamiento Punto Fijo
N_suavizado <- 2
suavizado_result <- suavizador_punto_fijo(x_filt, P_filt, gamma, p, Q, R,
N_suavizado)
x_smooth <- suavizado_result$x_smooth
P_smooth <- suavizado_result$P_smooth

# Cálculo de errores
e_filt <- x_filt - x_true
e_suav <- x_smooth - x_true

# Cálculo de varianzas
e_filt2 <- rep(0, K)
e_suav2 <- rep(0, K)

for (k in 1:K) {
  e_filt2[k] <- var(e_filt[1:k])
  e_suav2[k] <- var(e_suav[1:k])
}

# Gráficas
par(mfrow = c(1, 1), mar = c(4, 4, 2, 1))  # Ajuste de márgenes
plot(1:K, x_true, type = 'b', col = 'blue', pch = 19, main = 'Proceso
Verdadero y Observaciones', xlab = 'k', ylab = 'x_true')
points(1:K, z, type = 'b', col = 'red', pch = 19)
lines(1:K, x_filt, col = 'green', lty = 2)
lines(1:K, x_smooth, col = 'purple', lty = 2)
legend("topright", legend = c("x(k)", "Observaciones", "Filtro",
"Suavizado"), col = c("blue", "red", "green", "purple"), lty = c(1, 1, 2,
2), cex=0.5)

plot(1:K, e_filt2, type = "l", col = "red", ylab = "Varianza",
     xlab = "Iteración", main = "Varianzas de errores de filtrado y
suavizamiento")
lines(1:K, e_suav2, col = "green")
legend("topright", legend = c("Filtrado", "Suavizamiento"), col =
```

```
c("red", "green"), lty = 1)
```



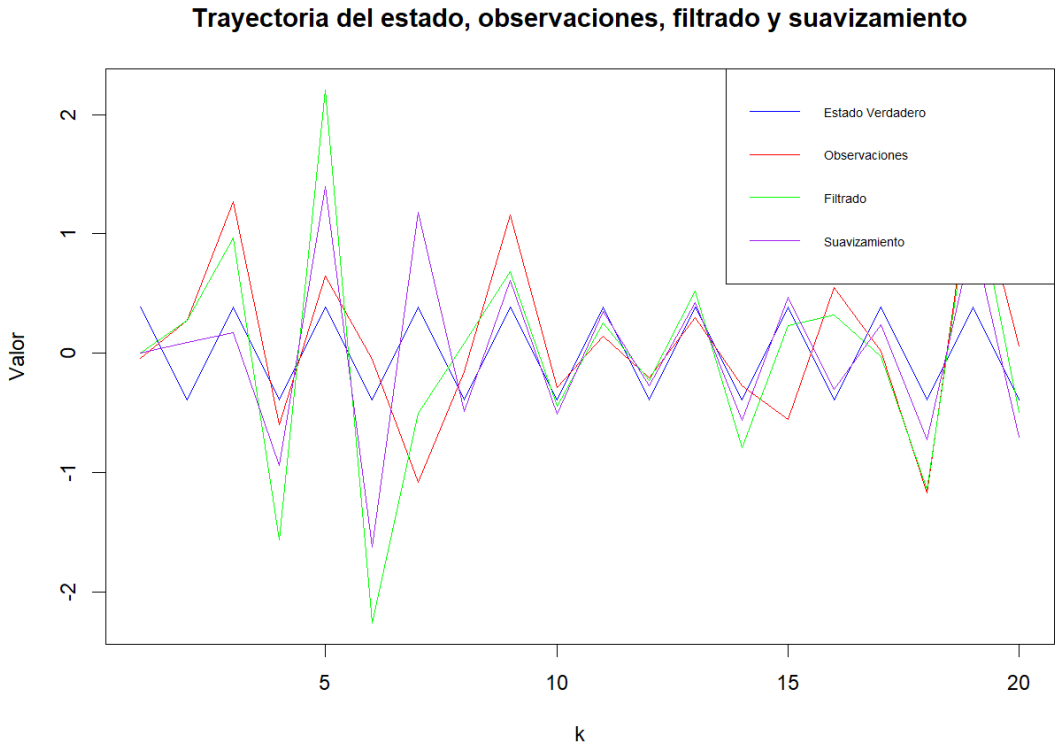


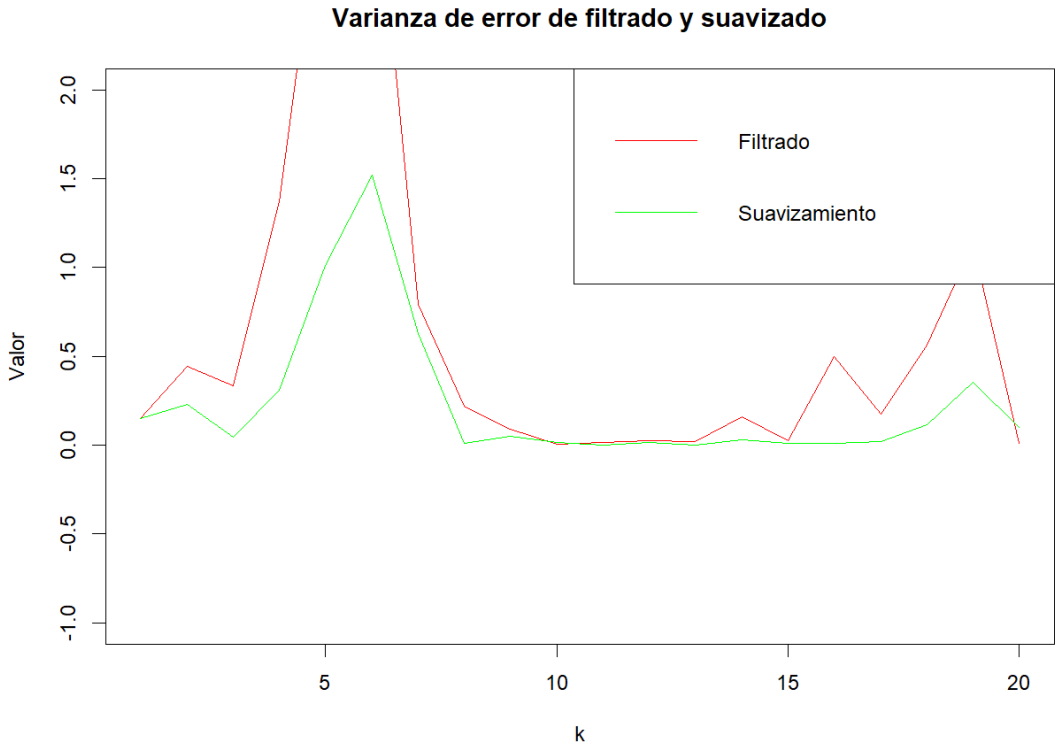
:

Apartado (c)

```
# Definir parámetros del sistema
N_iter <- 20  # Número de iteraciones
p <- 0.5      # Probabilidad del ruido multiplicativo
Q <- 0.5      # Varianza del ruido de proceso
R <- 0.5      # Varianza del ruido de medida
N_suav <- 2   # N para el suavizamiento

# Inicializar vectores y matrices
x <- numeric(N_iter)      # Estado real
z <- numeric(N_iter)      # Observaciones
x_hat_filt <- numeric(N_iter)  # Estimaciones de filtrado
x_hat_suav <- numeric(N_iter)  # Estimaciones de suavizamiento
P_filt <- numeric(N_iter)      # Varianzas de filtrado
P_suav <- numeric(N_iter)      # Varianzas de suavizamiento
K <- numeric(N_iter)       # Ganancia de Kalman
M <- numeric(N_iter)       # Ganancia de suavizamiento

# Condiciones iniciales
x[1] <- rnorm(1)  # Variable gaussiana con media cero y varianza 1
gamma <- rbinom(N_iter, size = 1, prob = p)
v <- rnorm(N_iter, mean = 0, sd = sqrt(Q))
z[1] <- gamma[1] * x[1] + v[1]
x_hat_filt[1] <- 0  # Estimación inicial
P_filt[1] <- 1
x_hat_suav[1] <- 0  # Estimación inicial
P_suav[1] <- 1

# Bucle for para filtrado y suavizamiento
for (k in 2:N_iter) {
  # Modelo del sistema
  phi <- (-1)^(2*k + 1)
  x[k] <- phi * x[k - 1]

  # Observación
  z[k] <- gamma[k] * x[k] + v[k]

  # Filtro de Kalman
  # Predicción
  x_hat_filt[k] <- phi * x_hat_filt[k - 1]
  P_filt[k] <- phi^2 * P_filt[k - 1] + Q

  # Ganancia de Kalman
  K[k] <- P_filt[k] / (P_filt[k] + gamma[k]^2 * R)

  # Actualización de la estimación y la covarianza
  x_hat_filt[k] <- x_hat_filt[k] + K[k] * (z[k] - gamma[k] *
x_hat_filt[k])
```

```
  P_filt[k] <- (1 - K[k] * gamma[k]) * P_filt[k]

  # Suavizador punto fijo
  if (k >= N_suav) {
    M[k] <- P_filt[k - N_suav + 1] * phi / (P_filt[k - N_suav + 1] *
phi^2 + Q)
    x_hat_suav[k] <- x_hat_filt[k] + M[k] * (x_hat_suav[k - 1] - phi *
x_hat_filt[k])
    P_suav[k] <- P_filt[k] - M[k]^2 * P_filt[k]
  }
}

# Gráfica de trayectoria del estado, observaciones, filtrado y
suavizamiento
# Trayectoria y observaciones
plot(1:N_iter, x, type = 'l', col = 'blue', ylim = c(min(x, z,
x_hat_filt, x_hat_suav), max(x, z, x_hat_filt, x_hat_suav)),
     ylab = 'Valor', xlab = 'k', main = 'Trayectoria del estado,
observaciones, filtrado y suavizamiento')
lines(1:N_iter, z, col = 'red')
lines(1:N_iter, x_hat_filt, col = 'green')
lines(1:N_iter, x_hat_suav, col = 'purple')
legend("topright", legend = c("Estado Verdadero", "Observaciones",
"Filtrado", "Suavizamiento"), col = c("blue", "red", "green", "purple"),
lty = 1, cex=0.6)

# Varianzas de errores de filtrado y suavizamiento
e_filt <- x - x_hat_filt
e_suav <- x - x_hat_suav
e_filt2 <- e_filt^2
e_suav2 <- e_suav^2

plot(1:N_iter, e_filt2, type = 'l', col = 'red', ylim = c(-1, 2),
     ylab = 'Valor', xlab = 'k', main = 'Varianza de error de filtrado y
suavizado')
lines(1:N_iter, e_suav2, col = 'green')
legend("topright", legend = c("Filtrado", "Suavizamiento"), col =
c("red", "green"), lty = 1)
```


Trayectoria del estado, observaciones, filtrado y suavizamiento
Estado Verdadero
Observaciones
Filtrado
Suavizamiento
Valor
k
-2
-1
0
1
2
5
10
15
20

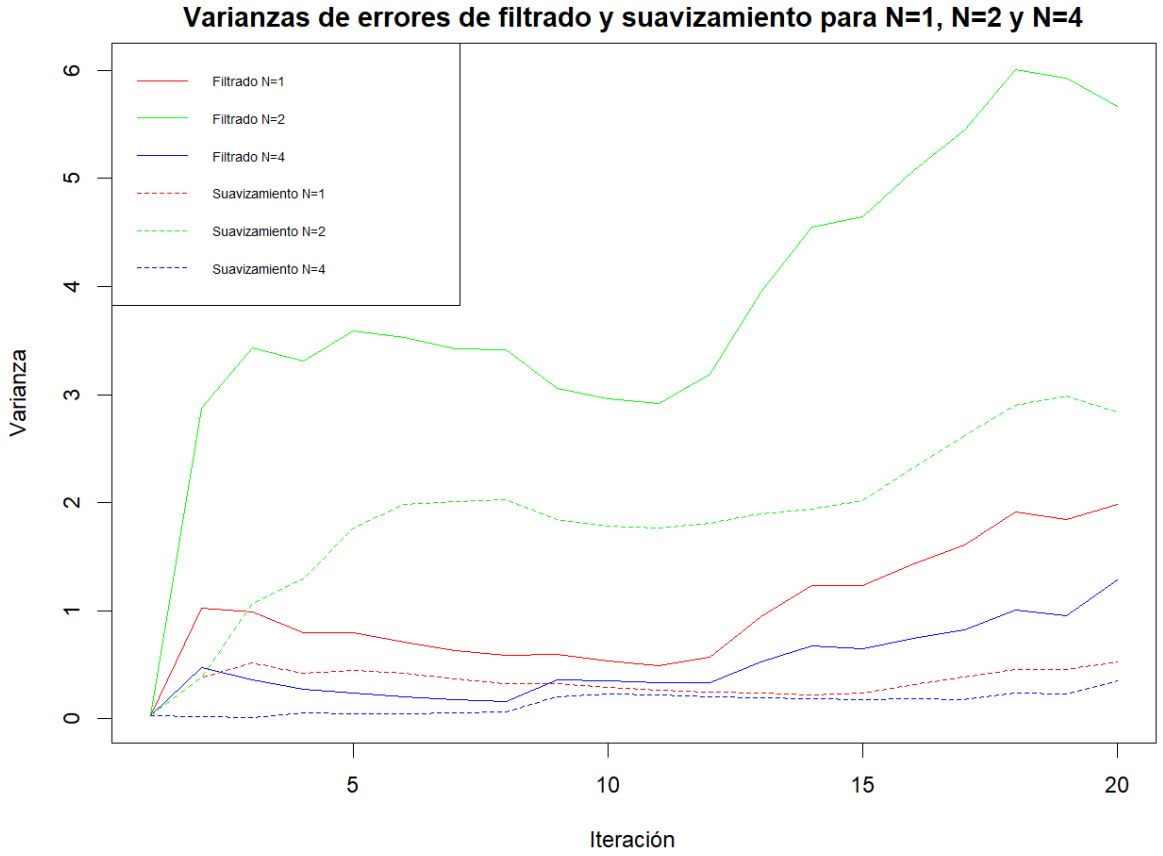

Varianza de error de filtrado y suavizado
Filtrado
Suavizamiento
Valor
k
-1.0
-0.5
0.0
0.5
1.0
1.5
2.0
5
10
15
20

Para representar las varianzas de errores de filtrado y suavizamiento con diferentes valores de N se edita el anterior script:

```
# Definir parámetros del sistema
N_iter <- 20  # Número de iteraciones
p <- 0.5      # Probabilidad del ruido multiplicativo
Q <- 0.5      # Varianza del ruido de proceso
R <- 0.5      # Varianza del ruido de medida
N_values <- c(1, 2, 4)  # Valores de N a probar

# Inicializar matrices para almacenar resultados
x <- numeric(N_iter)           # Estado real
z <- numeric(N_iter)           # Observaciones
x_filt <- matrix(0, nrow = N_iter, ncol = length(N_values))  #
Estimaciones de filtrado para diferentes N
x_smooth <- matrix(0, nrow = N_iter, ncol = length(N_values))  #
Estimaciones de suavizamiento para diferentes N
P_filt <- matrix(0, nrow = N_iter, ncol = length(N_values))  # Varianzas
de filtrado para diferentes N
P_smooth <- matrix(0, nrow = N_iter, ncol = length(N_values))  #
Varianzas de suavizamiento para diferentes N
K <- matrix(0, nrow = N_iter, ncol = length(N_values))  # Ganancias de
Kalman para diferentes N
M <- matrix(0, nrow = N_iter, ncol = length(N_values))  # Ganancias de
suavizamiento para diferentes N

# Generar observaciones inciertas
set.seed(100)
w <- rnorm(N_iter, mean = 0, sd = sqrt(Q))
gamma <- rbinom(N_iter, size = 1, prob = p)
v <- rnorm(N_iter, mean = 0, sd = sqrt(R))

# Aplicar el ciclo computacional del filtro y el suavizador para
diferentes N
for (N_index in 1:length(N_values)) {
  N <- N_values[N_index]

  # Inicializar vectores y matrices para el valor actual de N
  x_hat_filt <- numeric(N_iter)  # Estimaciones de filtrado para el valor
actual de N
  x_hat_suav <- numeric(N_iter)  # Estimaciones de suavizamiento para el
valor actual de N
  P_hat_filt <- numeric(N_iter)  # Varianzas de filtrado para el valor
actual de N
  P_hat_suav <- numeric(N_iter)  # Varianzas de suavizamiento para el
valor actual de N

  x[1] <- rnorm(1)
  z[1] <- gamma[1] * x[1] + v[1]
```

```
  x_hat_filt[1] <- 0  # Estimación inicial para el valor actual de N
  P_hat_filt[1] <- 1

  # Bucle for para filtrado y suavizamiento
  for (k in 2:N_iter) {
    # Modelo del sistema
    phi <- (-1)^(2*k + 1)
    x[k] <- phi * x[k - 1] + w[k]

    # Observación
    z[k] <- gamma[k] * x[k] + v[k]

    # Filtro de Kalman
    # Predicción
    x_hat_filt[k] <- phi * x_hat_filt[k - 1]
    P_hat_filt[k] <- phi^2 * P_hat_filt[k - 1] + Q

    # Ganancia de Kalman
    K[k, N_index] <- P_hat_filt[k] / (P_hat_filt[k] + gamma[k]^2 * R)

    # Actualización de la estimación y la covarianza
    x_hat_filt[k] <- x_hat_filt[k] + K[k, N_index] * (z[k] - gamma[k] *
x_hat_filt[k])
    P_hat_filt[k] <- (1 - K[k, N_index] * gamma[k]) * P_hat_filt[k]

    # Suavizador punto fijo
    if (k >= N) {
      M[k, N_index] <- P_hat_filt[k - N + 1] * phi / (P_hat_filt[k - N +
1] * phi^2 + Q)
      x_hat_suav[k] <- x_hat_filt[k] + M[k, N_index] * (x_hat_suav[k - 1]
- phi * x_hat_filt[k])
      P_hat_suav[k] <- P_hat_filt[k] + M[k, N_index]^2 * (P_hat_suav[k -
1] - P_hat_filt[k])
    }
  }

  # Almacenar resultados para el valor actual de N
  x_filt[, N_index] <- x_hat_filt
  x_smooth[, N_index] <- x_hat_suav
  P_filt[, N_index] <- P_hat_filt
  P_smooth[, N_index] <- P_hat_suav
}

# Cálculo de errores
e_filt <- x_filt - x
e_suav <- x_smooth - x

# Cálculo de varianzas
e_filt_var <- apply(e_filt^2, 2, cumsum) / (1:N_iter)
```

```r
e_suav_var <- apply(e_suav^2, 2, cumsum) / (1:N_iter)

# Gráfica: Varianzas de los errores de filtrado y suavizamiento para N=1, N=2 y N=4
par(mfrow = c(1, 1))
par(mar = c(4, 4, 2, 1))  # Ajustar los márgenes

plot(1:N_iter, e_filt_var[, 1], type = "l", col = "red", lty = 1, ylim=c(min(e_filt_var, e_suav_var), max(e_filt_var, e_suav_var)), ylab = "Varianza",
     xlab = "Iteración", main = "Varianzas de errores de filtrado y suavizamiento para N=1, N=2 y N=4")
lines(1:N_iter, e_filt_var[, 2], col = "green", lty = 1)
lines(1:N_iter, e_filt_var[, 3], col = "blue", lty = 1)
lines(1:N_iter, e_suav_var[, 1], col = "red", lty = 2)
lines(1:N_iter, e_suav_var[, 2], col = "green", lty = 2)
lines(1:N_iter, e_suav_var[, 3], col = "blue", lty = 2)
legend("topleft", legend = c("Filtrado N=1", "Filtrado N=2", "Filtrado N=4", "Suavizamiento N=1", "Suavizamiento N=2", "Suavizamiento N=4"),
       col = c("red", "green", "blue", "red", "green", "blue"), lty = c(1, 1, 1, 2, 2, 2), cex = 0.6)
```



Apartado (d)

```r
# Definir parámetros del sistema
n <- 1       # Dimensión del estado
N <- 50      # Número de iteraciones
x0 <- rnorm(1, mean = 0, sd = 1)     # Media del estado inicial
P0 <- 1      # Varianza del estado inicial
phi <- (-1)^(2*(1:N)+1)  # Coeficiente del modelo del sistema
Q <- 0.1     # Varianza del ruido de proceso
p_values <- c(0.1, 0.3, 0.5, 0.7, 0.9)  # Valores de probabilidad del
ruido multiplicativo
R <- 0.5     # Varianza del ruido de medida

# Inicializar vectores y matrices
x <- numeric(N)  # Estado real
z <- numeric(N)  # Observaciones
e_filt2_values <- matrix(0, nrow = N, ncol = length(p_values))  #
Varianzas de errores de filtrado

# Aplicar el ciclo computacional del filtro para diferentes valores de p
for (p_index in seq_along(p_values)) {
  p <- p_values[p_index]

  # Inicializar vectores y matrices
  x_filt <- numeric(N)  # Estimaciones de filtrado
  P_filt <- numeric(N)  # Varianzas de filtrado
  K <- numeric(N)  # Ganancia de Kalman

  # Generar observaciones inciertas
  set.seed(5)
  w <- rnorm(N, mean = 0, sd = sqrt(Q))
  gamma <- rbinom(N, size = 1, prob = p)
  v <- rnorm(N, mean = 0, sd = sqrt(R))
  x[1] <- x0 + w[1]
  z[1] <- gamma[1] * x[1] + v[1]

  # Aplicar el ciclo computacional del filtro
  for (k in 2:N) {
    # Modelo del sistema
    x[k] <- (-1)^(2*k+1) * x[k-1] + w[k]
    z[k] <- gamma[k] * x[k] + v[k]

    # Filtro de Kalman
    # Predicción
    x_filt[k] <- (-1)^(2*k+1) * x_filt[k-1]
    P_filt[k] <- ((-1)^(2*k+1))^2 * P_filt[k-1] + Q
    # Actualización
    K[k] <- P_filt[k] / (P_filt[k] + gamma[k]^2 * R)
    x_filt[k] <- x_filt[k] + K[k] * (z[k] - gamma[k] * x_filt[k])
```

```
    P_filt[k] <- (1 - K[k] * gamma[k]) * P_filt[k]

    # Cálculo de los errores de filtrado y su varianza
    e_filt <- x_filt - x
    e_filt2_values[k, p_index] <- var(e_filt)
  }
}

# Graficar las varianzas de los errores de filtrado para diferentes
valores de p
plot(1:N, e_filt2_values[, 1], type = "l", col = "red", ylim = c(0,
max(e_filt2_values)),
     ylab = "Varianza del error de filtrado", xlab = "Iteración", main =
"Varianzas de errores de filtrado para distintos valores de p")
for (i in 2:length(p_values)) {
  lines(1:N, e_filt2_values[, i], col = rainbow(length(p_values))[i])
}
legend("topleft", legend = c(p_values), col = rainbow(length(p_values)),
lty = 1)
```