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 $\Pi(k) = E[ze(k/k-1)ze^T(k/k-1)]$ 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 $P(k/k-1) = E[x_e(k/k-1)x_e^T(k/k-1)]$ 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 $k = 0$:

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

Para el caso general ($k > 0$), se puede aplicar la ley de las esperanzas totales:

$$\tilde{z}(k/k-1) = z(k) - E[\tilde{z}(k/k-1)|z(k-1)] = z(k) - p(k)H(k)\hat{x}(k/k-1)$$

donde $\hat{x}(k/k-1)$ es la estimación del estado en el instante k dado $z(k-1)$.

Si calculamos y desarrollamos $\Pi(k)$, y aplicamos las definiciones de $D(k)$ y $Q(k-1)$.

$$\begin{aligned}\Pi(k) &= E[\tilde{z}(k/k-1)\tilde{z}T(k/k-1)] \\ &= E\left[\left(z(k) - p(k)H(k)\hat{x}(k/k-1)\right)\left(z(k) - p(k)H(k)\hat{x}(k/k-1)\right)^T\right] \\ &= R(k) - p(k)H(k)\Phi(k,k-1)E[x(k-1)] + p^2(k)H(k)\Phi(k,k-1)D(k-1)\Phi^T(k,k-1)H^T(k) + p^2(k)H(k)Q(k-1)H^T(k) \\ &= R(k) - p(k)H(k)\Phi(k,k-1)D(k-1)H^T(k)\end{aligned}$$

Y cómo pasas a la última igualdad

Y por lo tanto, se obtiene la definición buscada para $\Pi(k)$:

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

Apartado (b)

Lema de Proyecciones Ortogonales:

Supongamos que tenemos un espacio vectorial $\mathbb{V}$ y dos subespacios cerrados $\mathbb{U}$ y $\mathbb{W}$ de $\mathbb{V}$, donde $\mathbb{U}$ es de dimensión finita. Sea $\mathbb{P}_{\mathbb{U}}$ la proyección ortogonal sobre $\mathbb{U}$ y $\mathbb{P}_{\mathbb{W}}$ la proyección ortogonal sobre $\mathbb{W}$.

Entonces, el Lema de Proyecciones Ortogonales establece que para cualquier vector $\mathbf{v} \in \mathbb{V}$, se cumple que:

$$\mathbf{v} = \mathbb{P}_{\mathbb{U}}(\mathbf{v}) + \mathbb{P}_{\mathbb{W}}(\mathbf{v})$$

y estas proyecciones son ortogonales entre sí. Es decir, $\mathbb{P}_{\mathbb{U}}(\mathbf{v})$ es la componente de $\mathbf{v}$ en $\mathbb{U}$ y $\mathbb{P}_{\mathbb{W}}(\mathbf{v})$ es la componente de $\mathbf{v}$ en $\mathbb{W}$, y son ortogonales entre sí.

Algoritmo Recursivo para el Problema de Suavizamiento Punto Fijo Lineal:

Dado el sistema con observaciones inciertas, el objetivo es deducir el algoritmo recursivo para el problema de suavizamiento punto fijo lineal utilizando el Lema de Proyecciones Ortogonales.

El problema de suavizamiento punto fijo lineal busca estimar el estado pasado $x(k)$ dado todas las observaciones pasadas $z(0), z(1), \ldots, z(K)$, donde $K \geq k$. Denotemos la estimación a posteriori de $x(k)$ dados $z(0), z(1), \ldots, z(K)$ como $\hat{x}(k|K)$.

La relación del filtro de Kalman es:

$$\hat{x}(k|k) = \hat{x}(k|k-1) + K(k)e(k)$$

donde $K(k)$ es la ganancia de Kalman en el instante k y $e(k)$ es la innovación en el instante k. Además, la estimación a posteriori de $x(k)$ dado $z(0), z(1), \ldots, z(k)$ es:

$$\hat{x}(k|k) = \hat{x}(k|k) + J(k)[\hat{x}(k+1|k) - \Phi(k+1,k)\hat{x}(k|k)]$$

donde $J(k)$ es la matriz de suavizamiento en el instante k.

Aplicamos el Lema de Proyecciones Ortogonales para descomponer $\hat{x}(k+1|k) - \Phi(k+1,k)\hat{x}(k|k)$ en dos componentes ortogonales, una en el espacio de las observaciones ($H(k)$) y otra en el espacio de los estados ($I - \Phi(k+1,k)$):

$$\begin{aligned}\hat{x}(k|k) &= \hat{x}(k|k) + J(k)[\hat{x}(k+1|k) - \Phi(k+1,k)\hat{x}(k|k)] \\ &= \hat{x}(k|k) + J(k)\left[\mathbb{P}_{H(k)}\left(\hat{x}(k+1|k) - \Phi(k+1,k)\hat{x}(k|k)\right) + \mathbb{P}_{I-\Phi(k+1,k)}\left(\hat{x}(k+1|k) - \Phi(k+1,k)\hat{x}(k|k)\right)\right]\end{aligned}$$

Entonces, gracias a las proyecciones ortogonales, el problema se descompone en dos partes: estimar la componente en el espacio de observaciones, y estimar la componente en el espacio de estados.

6. Componente en el espacio de las observaciones:

No entiendo de dónde obtienes esto

$$\hat{x}_o(k|k) = \hat{x}(k|k) + J(k)\mathbb{P}_{H(k)}\left(\hat{x}(k+1|k) - \Phi(k+1,k)\hat{x}(k|k)\right)$$

2. Componente en el espacio de los estados:

$$\hat{x}_s(k|k) = \hat{x}(k|k) + J(k)\mathbb{P}_{I-\Phi(k+1,k)}\left(\hat{x}(k+1|k) - \Phi(k+1,k)\hat{x}(k|k)\right)$$

Estos subproblemas pueden resolverse recursivamente para obtener el algoritmo completo de suavizamiento punto fijo lineal. La matriz de suavizamiento $J(k)$ y las ganancias de Kalman $K(k)$ se actualizan en cada paso del filtro hacia atrás. La ganancia de Kalman se calcula utilizando la ecuación del filtro de Kalman:

$$K(k) = P(k|k-1)\Phi^T(k,k-1)[\Phi(k,k-1)P(k|k-1)\Phi^T(k,k-1) + Q(k-1)]^{-1}$$

y la matriz de suavizamiento se calcula como:

$$J(k) = P(k|k-1)\Phi^T(k,k-1)[\Phi(k,k-1)P(k|k-1)\Phi^T(k,k-1) + Q(k-1)]^{-1}$$

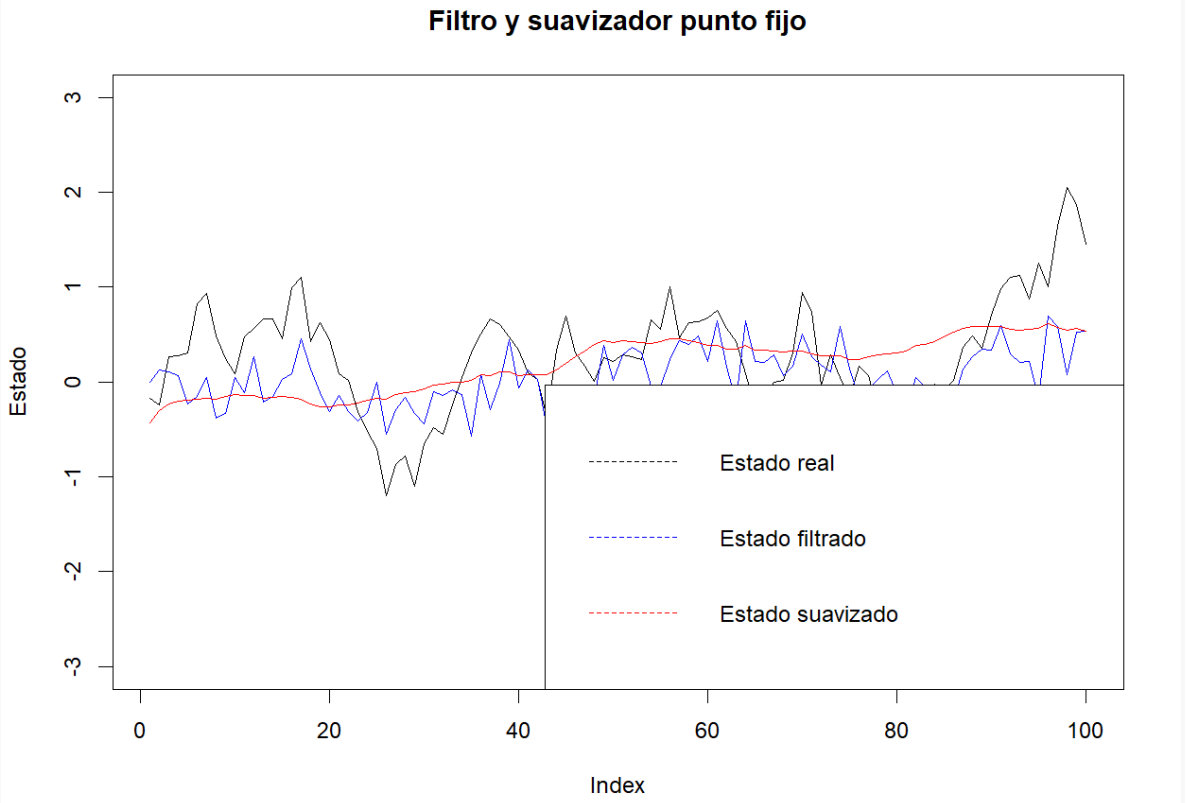
Estas actualizaciones se realizan recursivamente desde $k = K$ hasta $k = 0$, utilizando las estimaciones a posteriori del filtro de Kalman ($\hat{x}(k|k)$) y las estimaciones a posteriori de suavizamiento ($\hat{x}(k|K)$) en cada paso.

Apartado (c)

```r
# 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
N <- 100
```

```r
# 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)
```

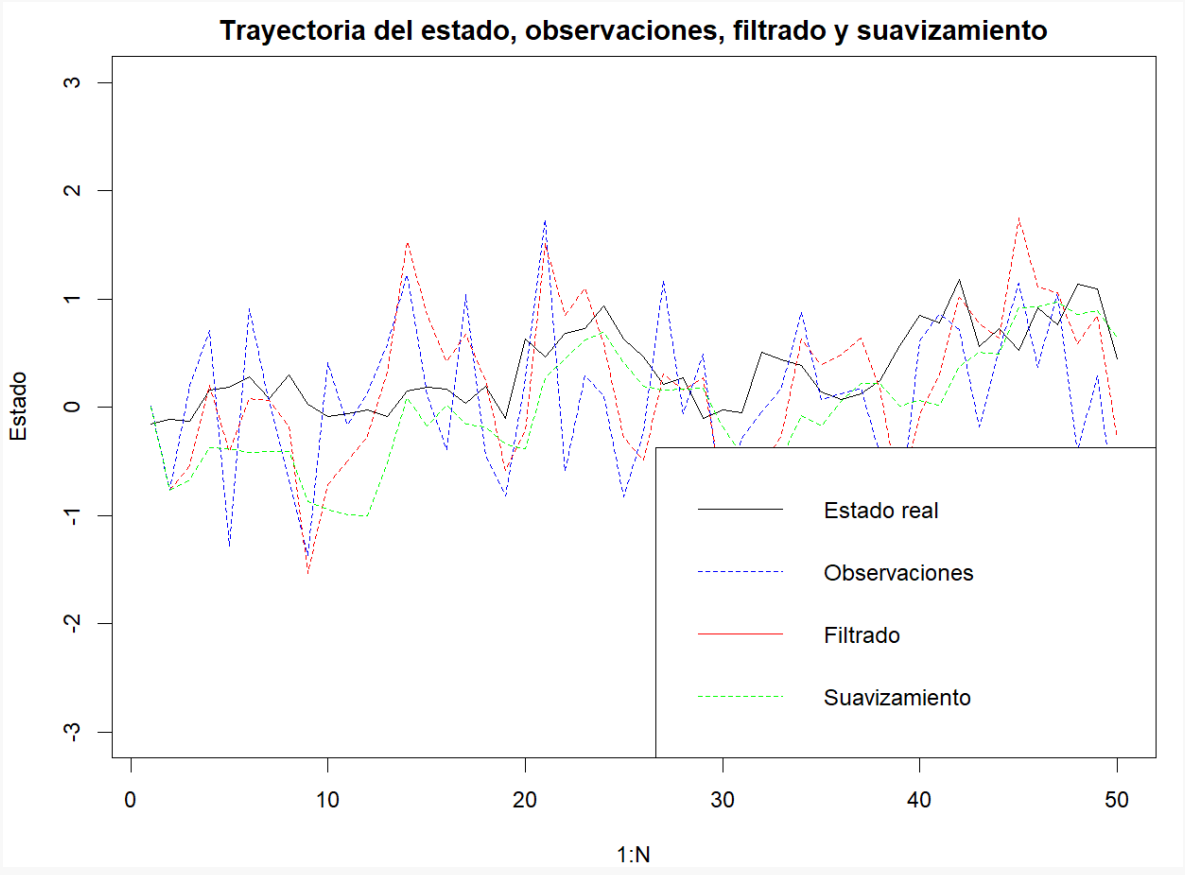


Apartado (d)

```
# 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
```

```r
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(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 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]
  }
}

# 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, P_filt, 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, P_smooth, col = "green")
legend("topright", legend = c("Filtrado", "Suavizamiento"), col =
c("red", "green"), lty = 1)
```

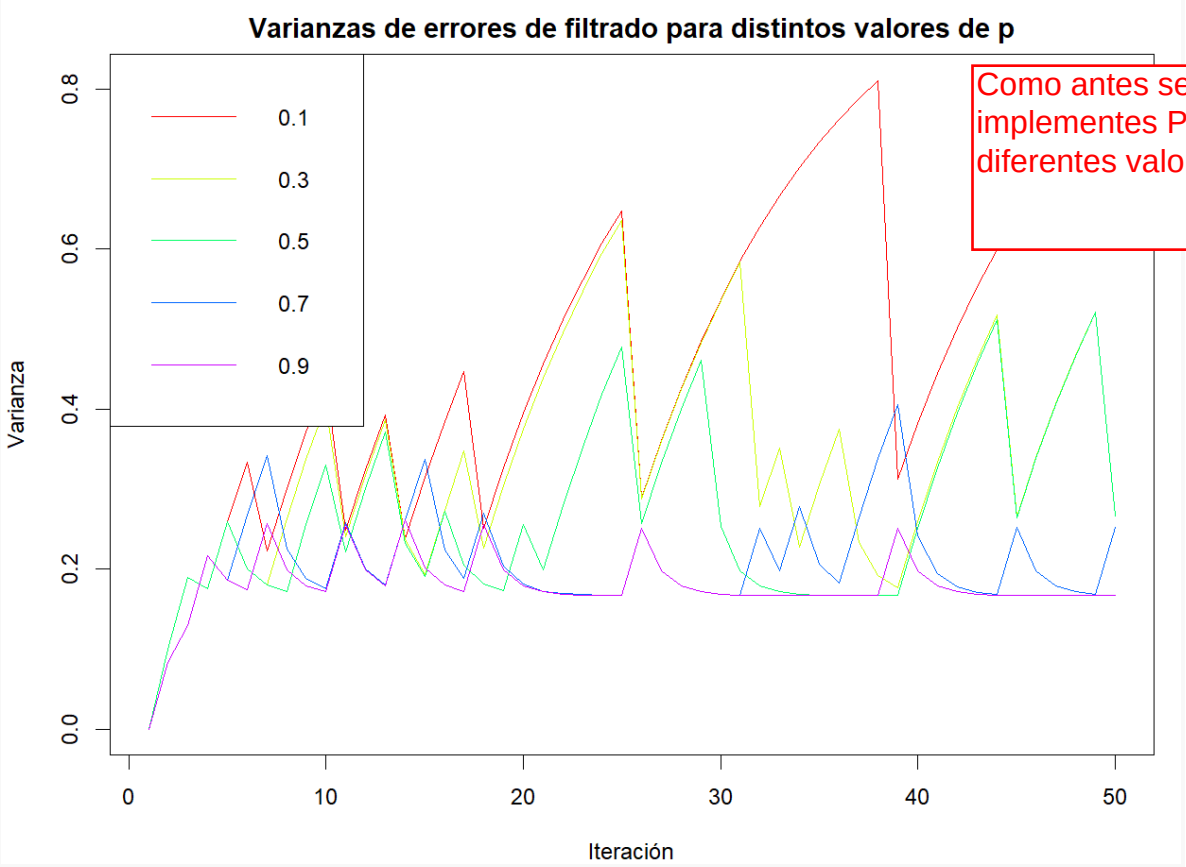


No te pueden salir las varianzas de suavizamiento mayores que las de filtrado por una parte y por otra lo que se pide es que implementes la expresión obtenida en el algoritmo. P(k/k) y P(k/j)

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
x_filt_values <- matrix(0, nrow = N, ncol = length(p_values))  #
Varianzas de filtrado

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

```r
  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]
  }

  # Almacenar los resultados para el valor específico de p
  x_filt_values[, p_index] <- P_filt
}

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



Como antes se pide que implementes P(k/k) para diferentes valores de p

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.

En todas se observa una gráfica con formas de dientes de sierra, indicando momentos en los que la incertidumbre aumenta y disminuye de manera abrupta. Esto se debe, principalmente, al ruido multiplicativo, que sigue una distribución de Bernoulli

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

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

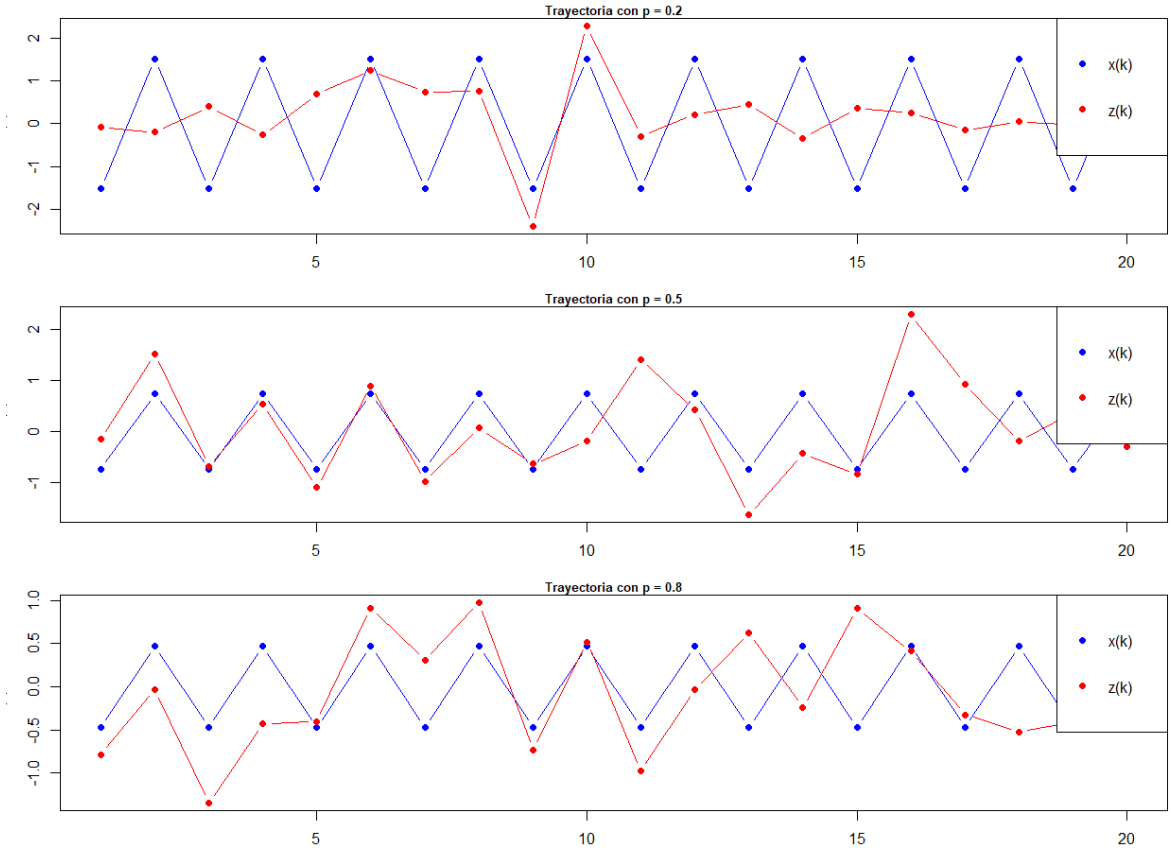
donde el ruido multiplicativo $\{\gamma(k); k \geq 0\}$ es una sucesión de variables aleatorias independientes de Bernoulli, con $P(\gamma(k) = 1) = p$, y el ruido aditivo $\{v(k); k \geq 0\}$ es una sucesión blanca gaussiana, centrada y con varianzas $E[v^2(k)] = 0.5, \quad k \geq 0$.

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)
}
```



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.

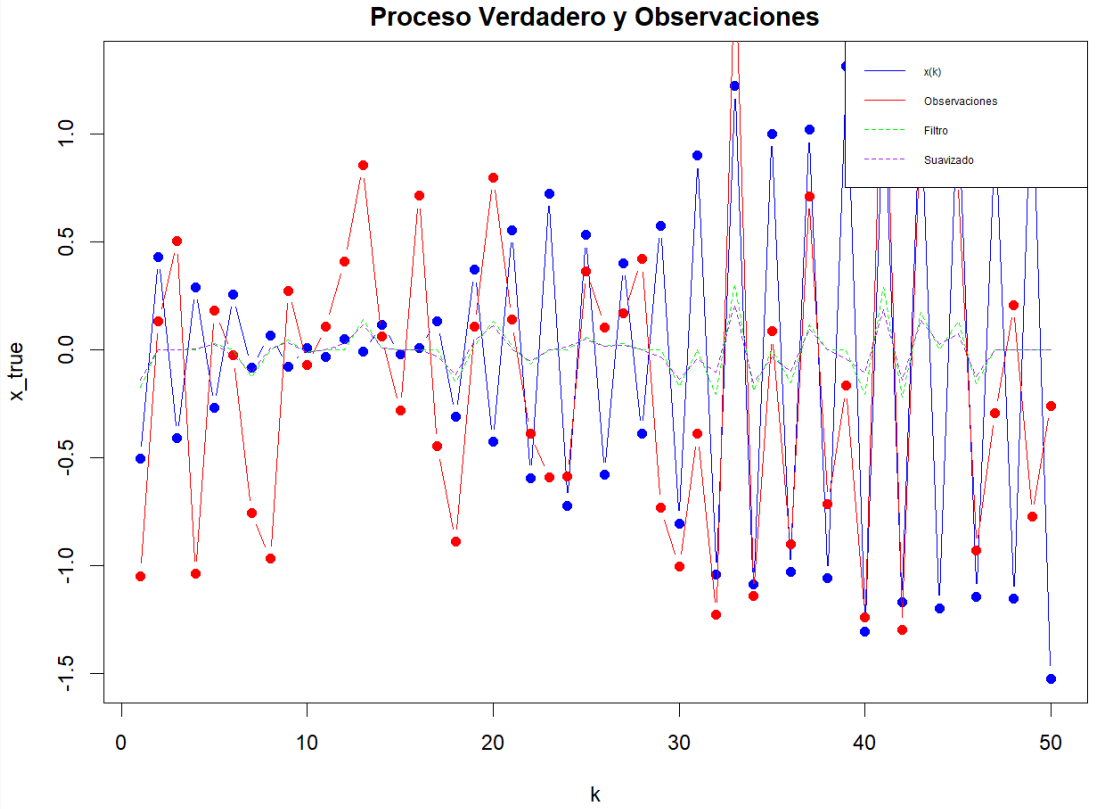
Las observaciones del proceso son ruidosas y dependen del valor del ruido multiplicativo, que es una variable de Bernoulli. Cuando el ruido multiplicativo es cero, la observación es igual al ruido aditivo, que es una variable gaussiana centrada. Cuando el ruido multiplicativo es uno, la observación es 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]  #
Modificación aquí
      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
set.seed(100)
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)
```

```r
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

# 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)

par(mfrow = c(1, 1), mar = c(4, 4, 2, 1))  # Ajuste de márgenes
plot(1:K, P_filt, type = 'b', col = 'blue', pch = 19, main = 'Varianzas
de Errores', xlab = 'k', ylab = 'Varianza')
points(1:K, P_smooth, col = 'red', pch = 19)
legend("topright", legend = c("Filtro", "Suavizado"), col = c("blue",
"red"), lty = c(1, 1))
```


Proceso Verdadero y Observaciones
x(k)
Observaciones
Filtro
Suavizado
x_true
1.0
0.5
0.0
-0.5
-1.0
-1.5
0
10
20
30
40
50
k


Varianzas de Errores
Filtro
Suavizado
Varianza
0.100
0.095
0.090
0.085
0
10
20
30
40
50
k
Igual que en los anteriores
para las varianzas

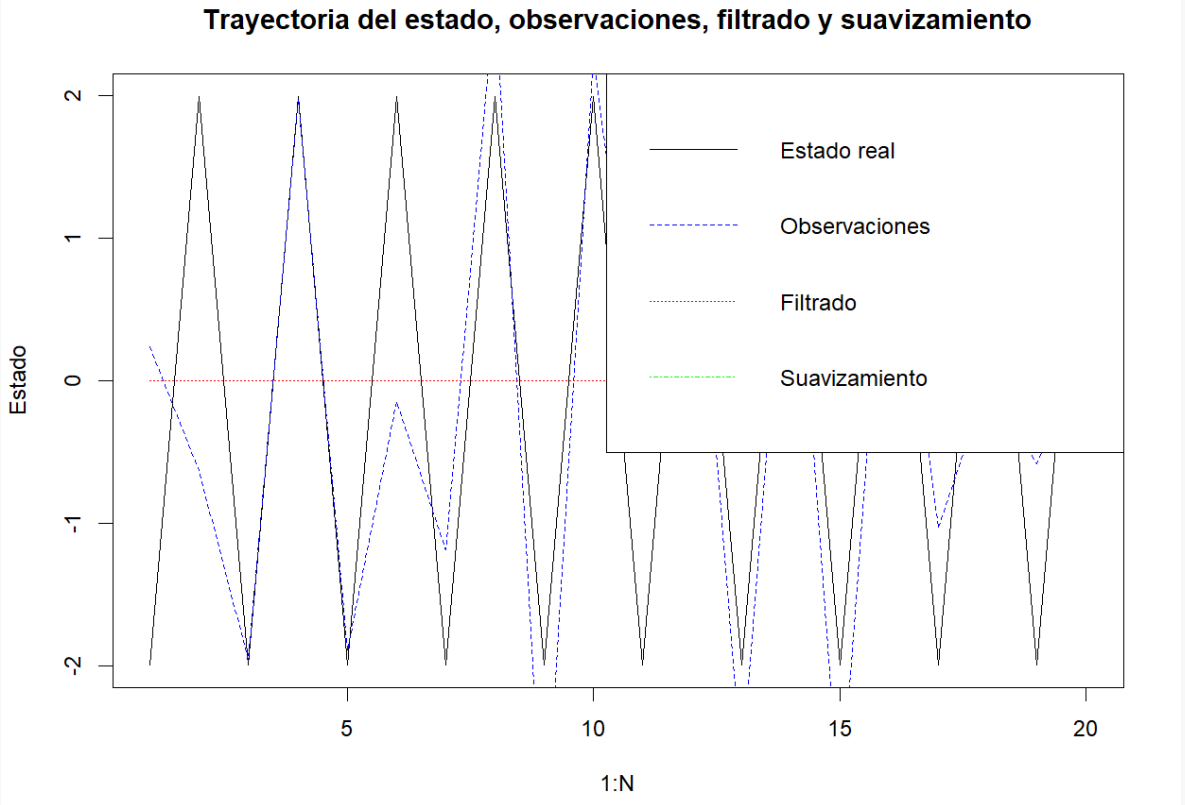
Este gráfico está hecho para N=2. La manera más eficiente sería definir un vector c(1,2,4) para hacer también N=1 y N=4 y posteriormente adaptar un bucle for para que vaya guardando los valores de las tres N.

Apartado (c)

```
# Definir parámetros del sistema
N <- 20      # Número de iteraciones
p <- 0.5     # Probabilidad del ruido multiplicativo
var_v <- 0.5 # Varianza del ruido aditivo

# Inicializar vectores y matrices
x <- numeric(N)  # Estado real
z <- numeric(N)  # Observaciones
x_filt_values <- numeric(N)  # Estimaciones de filtrado
x_smooth_values <- numeric(N)  # Estimaciones de suavizamiento
P_filt_values <- numeric(N)  # Varianzas de filtrado
P_smooth_values <- numeric(N)  # Varianzas de suavizamiento

# Aplicar el ciclo computacional del filtro y suavizador
x[1] <- rnorm(1, mean = 0, sd = 1)  # Inicializar el estado real
z[1] <- rbinom(1, size = 1, prob = p) * x[1] + rnorm(1, mean = 0, sd =
sqrt(var_v))  # Inicializar la observación
for (k in 2:N) {
  x[k] <- (-1)^(2*k+1) * x[k-1]  # Modelo del sistema
  z[k] <- rbinom(1, size = 1, prob = p) * x[k] + rnorm(1, mean = 0, sd =
sqrt(var_v))  # Observación

  # Filtrado de Kalman
  # Predicción
  x_filt_values[k] <- (-1)^(2*k+1) * x_filt_values[k-1]
  P_filt_values[k] <- abs(x_filt_values[k-1])
  # Actualización
  K <- P_filt_values[k] / (P_filt_values[k] + var_v)
  x_filt_values[k] <- x_filt_values[k] + K * (z[k] - x_filt_values[k])
  P_filt_values[k] <- (1 - K) * P_filt_values[k]
}

# Suavizamiento de Kalman
x_smooth_values[N] <- x_filt_values[N]
P_smooth_values[N] <- P_filt_values[N]
for (k in (N-1):1) {
  J <- P_filt_values[k] / P_filt_values[k+1]
  x_smooth_values[k] <- x_filt_values[k] + J * (x_smooth_values[k+1] -
x_filt_values[k])
  P_smooth_values[k] <- P_filt_values[k] + J^2 * (P_smooth_values[k+1] -
P_filt_values[k+1])
```

```
}

# Graficar trayectoria del estado, observaciones, filtrado y
suavizamiento
plot(1:N, x, type = "l", col = "black", ylim = range(c(x, x_filt_values,
x_smooth_values), finite = TRUE),
     ylab = "Estado", main = "Trayectoria del estado, observaciones,
filtrado y suavizamiento")
lines(1:N, z, col = "blue", lty = 2)
lines(1:N, x_filt_values, col = "red", lty = 3)
lines(1:N, x_smooth_values, col = "green", lty = 4)
legend("topright", legend = c("Estado real", "Observaciones", "Filtrado",
"Suavizamiento"), col = c("black", "blue", "red", "green"), lty = c(1, 2,
3, 4))

# Graficar varianzas de errores de filtrado y suavizamiento
N_values <- c(1, 2, 4)
P_filt_values_N <- matrix(0, nrow = N, ncol = length(N_values))
P_smooth_values_N <- matrix(0, nrow = N, ncol = length(N_values))
for (N_val in N_values) {
  x_filt <- numeric(N)  # Estimaciones de filtrado
  P_filt <- numeric(N)  # Varianzas de filtrado
  K <- numeric(N)  # Ganancia de Kalman

  # Aplicar el ciclo computacional del filtro
  x_filt[1] <- rnorm(1, mean = 0, sd = 1)  # Inicializar el estado
filtrado
  P_filt[1] <- abs(x_filt[1])
  for (k in 2:N) {
    # Predicción
    x_filt[k] <- (-1)^(2*k+1) * x_filt[k-1]
    P_filt[k] <- abs(x_filt[k-1])
    # Actualización
    K[k] <- P_filt[k] / (P_filt[k] + var_v)
    x_filt[k] <- x_filt[k] + K[k] * (z[k] - x_filt[k])
    P_filt[k] <- (1 - K[k]) * P_filt[k]
  }

  # Suavizamiento
  x_smooth_update <- x_filt[N] + P_filt[N] * (x_smooth_values[N] -
x_filt[N])
  x_smooth <- numeric(N)
  P_smooth <- numeric(N)
  x_smooth[N] <- x_smooth_update
  P_smooth[N] <- P_filt[N] + P_smooth_values[N] * (P_smooth[N] -
P_filt[N])
  for (k in (N-1):1) {
    # Suavización
```

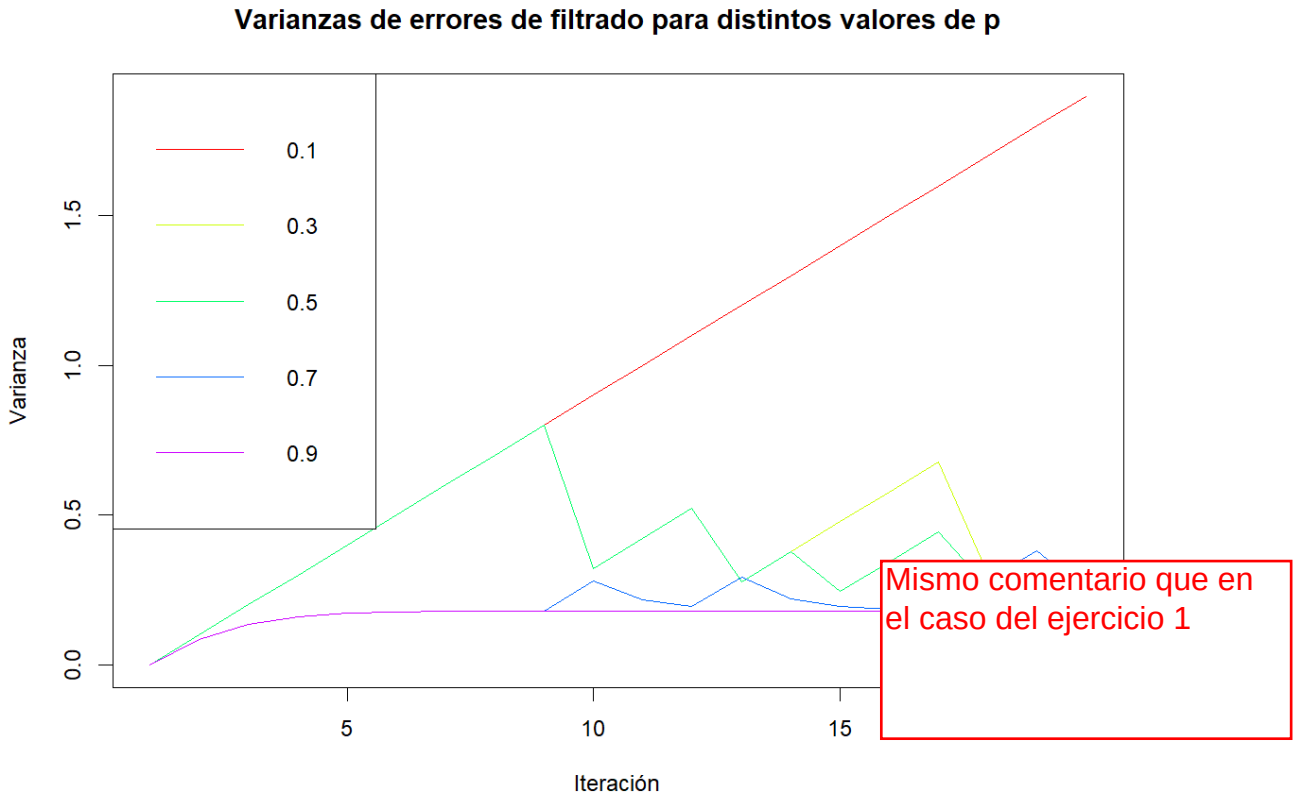
```
    J <- P_filt[k] / P_filt[k+1]
    x_smooth_update <- x_filt[k] + P_filt[k] * J * (x_smooth_update -
x_filt[k])
    x_smooth[k] <- x_smooth_update
    P_smooth[k] <- P_filt[k] + P_filt[k] * J^2 * (P_smooth[k+1] -
P_filt[k+1])
  }

  # Almacenar los resultados para el valor específico de N
  P_filt_values_N[, N_val] <- P_filt
  P_smooth_values_N[, N_val] <- P_smooth + N_val
}

# Graficar varianzas de errores de filtrado y suavizamiento para
diferentes valores de N
plot(1:N, P_filt_values_N[, 1], type = "l", col = "red", ylim =
range(P_filt_values_N, P_smooth_values_N, finite = TRUE),
     ylab = "Varianza", main = "Varianzas de errores de filtrado y
suavizamiento")
lines(1:N, P_smooth_values_N[, 1], col = "green")
for (i in 2:length(N_values)) {
  lines(1:N, P_filt_values_N[, ii], col = rainbow(length(N_values))[i],
lty = i)
  lines(1:N, P_smooth_values_N[, i], col = rainbow(length(N_values))[i],
lty = i)
}
legend("topright", legend = c(paste("Filtrado N =", N_values[1]),
paste("Suavizamiento N =", N_values[1]),
                              paste("Filtrado N =", N_values[2]),
paste("Suavizamiento N =", N_values[2]),
                              paste("Filtrado N =", N_values[3]),
paste("Suavizamiento N =", N_values[3])),
       col = c("red", "green", rainbow(length(N_values))[1],
rainbow(length(N_values))[1], rainbow(length(N_values))[2],
rainbow(length(N_values))[2]),
       lty = c(1, 1, 2, 2, 3, 3))
```



Apartado (d)

```
# Definir parámetros del sistema
n <- 1       # Dimensión del estado
N <- 20      # 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
x_filt_values <- matrix(0, nrow = N, ncol = length(p_values))  #
Varianzas 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]
```

```r
  # 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[k] * x[k-1] + w[k]
    z[k] <- gamma[k] * x[k] + v[k]

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

  # Almacenar los resultados para el valor específico de p
  x_filt_values[, p_index] <- P_filt
}

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



Mismo comentario que en el caso del ejercicio 1

Cuando p es pequeño (como en el caso de p=0.1), el término es menor en magnitud, lo que lleva a un aumento en la ganancia de Kalman. A medida que el denominador se convierte en una parte más significativa, la varianza de filtrado tiende a aumentar linealmente.

Por otro lado, cuando p es grande (como en el caso de p=0.9), el término es más grande en magnitud, lo que lleva a una disminución en la ganancia de Kalman. Esto puede resultar en una menor influencia de las observaciones y, por lo tanto, una menor varianza de filtrado.

Se puede ver el cambio de tendencia del filtrado al cambiar el parámetro, suavizándose las curvas cuando aumenta R. Además, se puede ver un cambio en el comportamiento de la función en relación a los parámetros.