Problema 1

A partir del algoritmo MH usando Kerneles Híbridos simule valores de la distribución normal bivariada, fijando $\sigma_1 = \sigma_2 = 1$, considere los casos $\rho = 0.8 y \rho = 0.99$.

Presentamos la implementación del algoritmo de Metropolis-Hastings con Kerneles Híbridos que utilizaremos en los 3 problemas de esta tarea. La función se llama **metropolis_hastings_hybrid_kernels()** y se encuentra implementada en el archivo **.py** del mismo nombre. Esta función recibe el tamaño de muestra requerido, la función de densidad posterior, un vector de densidades de las distintas propuestas y un vector de probabilidades discretas para escoger los diferentes kernels.

Algorithm 1: Metropolis-Hastings con Kerneles Híbridos. Extracto de **metropolis_hastings_hybrid_kernels.py**

```
# Sample until desired number of samples generated
while len(sample) < sample_size:

    # Select transition kernel
    idx = np.random.choice(num_of_proposals, p=kernel_probs)

    # Generate random step from proposed distribution
    x_p = step()

    # Calculate the acceptance ratio
    alpha = log_acceptance_ratio(x_p, x_t, log_posterior[idx], log_proposal[idx])

    # Random uniform sample
    u = np.random.uniform()

    if u < np.exp(alpha):  # Accept
        x_t = x_p

        # Burn-in
        ...

    # Add sample
    sample.append(x_t)

    # Next step
    t += 1
```

Una de las cosas que debemos mencionar es que en esta implementación del algoritmo de Metropolis-Hastings primero calculamos analíticamente (o al menos intentando minimizar el error numerico) el logaritmo de la razón de aceptación, osea que en vez de calcular,

$$\rho = min\left\{1, \frac{\pi(y)}{\pi(x)}\frac{q(x|y)}{q(y|x)}\right\}$$

calcularemos su logaritmo, notando que estamos aplicando una función creciente :

$$\rho' = \log(\rho) = min\left\{0, \log[\pi(y)] - \log[\pi(x)] + \log[q(x|y)] - \log[q(y|x)]\right\}$$

Como vimos en clase esto tiene mejores propiedades para hacer calculos numericos. Al final no aceptamos un paso propuesto con propabilidad $exp(\rho')$, osea que generamos un valor $u \sim \mathcal{U}(0, 1)$ y aceptamos cuando $u < exp(\rho')$.

En este problema queremos generar una muestra de la función objetivo del enunciado.Implementamos la función de distribución posterior (objetivo) en nuestro código de python de la siguiente manera :

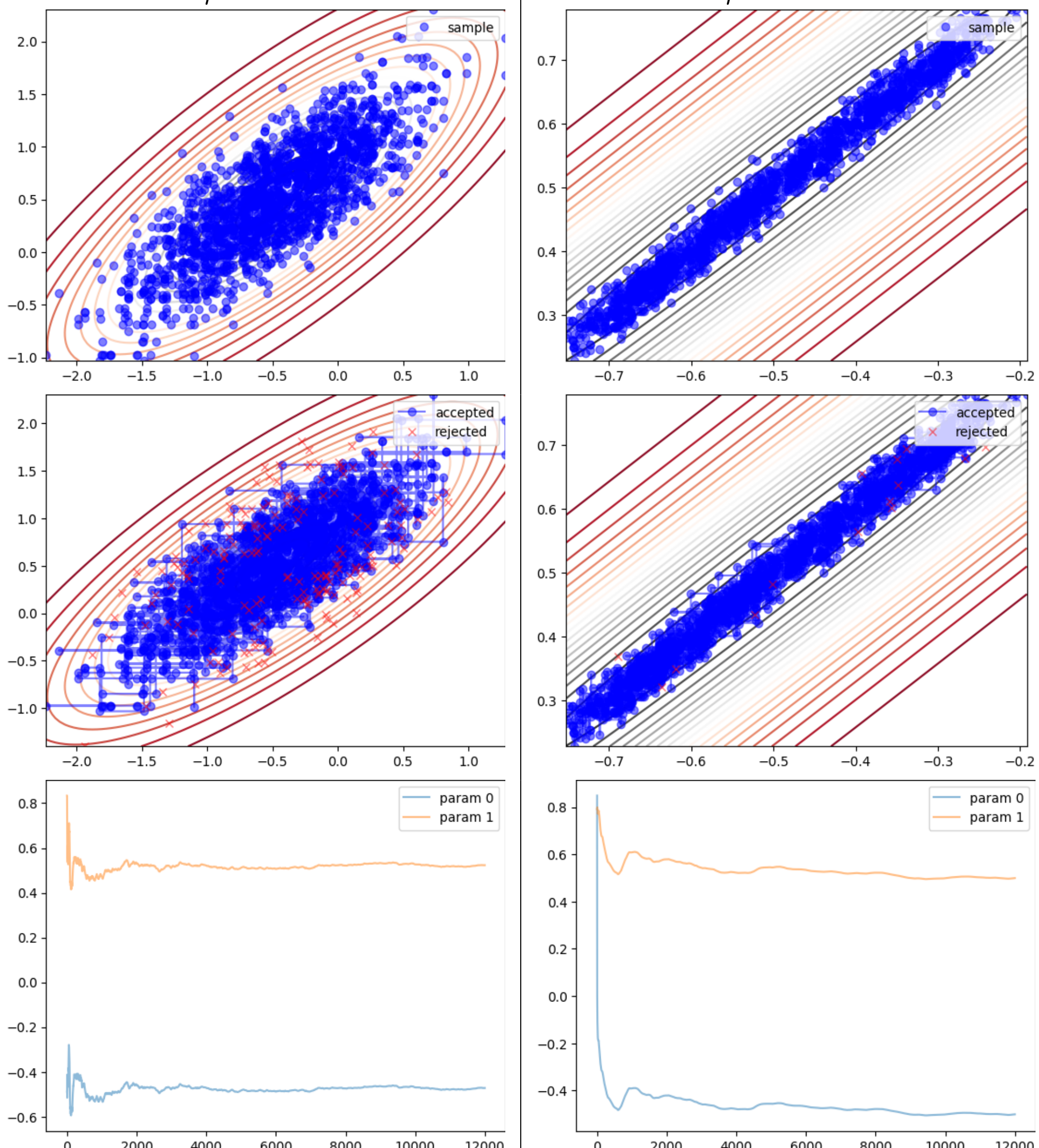
Algorithm 2: Posterior Normal

```python
# Posterior distribution params
mu_1 = -.5
mu_2 = .5
sigma_1 = sigma_2 = 1

# Construct vectors
MU = [mu_1, mu_2]
SIGMA = [[sigma_1 ** 2, rho * sigma_1 * sigma_2],
         [rho * sigma_1 * sigma_2, sigma_2 ** 2]]

# Posterior to sample with metropolis hastings
def posterior(x):
    return stats.multivariate_normal.pdf(x, MU, SIGMA)

def log_posterior(x):
    return stats.multivariate_normal.logpdf(x, MU, SIGMA)
```

Escogemos específicamente los parámetros $\mu_1 = -.5, \mu_2 = .5, \sigma_1 = \sigma_2 = 1$. Ejecutamos nuesto algoritmo para los distinos valores $\rho = 0.8$ y $\rho = 0.99$. Se escogio al azar en cada paso la propuesta a utilizar y se generó una muestra de tamaño 2,000 con un burn-in de 10,000 para observar la convergencia. Como posición inicial de nuestro algorimto utilizamos un valor generado con distribución $\mathcal{U}(-1, 1)$ en cada entrada.

A continuación estudiemos dos ejecuciones de nuestro algoritmo :

$\boldsymbol{\rho = 0.8}$ $\boldsymbol{\rho = 0.99}$



X1 : KS-Test P-Value = 1.1486-e229
X2 : KS-Test P-Value = 1.7224-e215

X1 : KS-Test P-Value = 0.0
X2 : KS-Test P-Value = 0.0

Podemos encontrar que ambas muestras 'siguen' las curvas de nivel de la distribución posterior. Esto lo podemos corroborar, un poco más, con una prueba de **Normalidad Kolmgorov-Smirnoff** de las muestras marginales X_1 y X_2, donde vemos que claramente se rechaza la hipotesis de que no sea normal. También notamos la grafica de convergencia de la media de la muestra de ambas entradas $x_1 \sim X_1, x_2 \sim X_2$ respecto al tiempo (número de iteraciones). Notamos que ambas medias estimadas $\hat{\mu_1}, \hat{\mu_2}$ convergen a los valores reales de las medias de las distribuciones marginales $\mu_1 = .5, \mu_2 = -.5$. Notamos que en el caso con el parámetro

Parámetros	**Usos**			**# Rechazados**			**% Rechazados**		
	P1	**P2**	**Tot.**	**P1**	**P2**	**Tot.**	**P1**	**P2**	**Tot.**
$\rho = 0.8$	1013	987	2000	113	92	205	11.15 %	9.32 %	10.25 %
$\rho = 0.99$	1015	985	2000	3	6	9	0.30 %	0.61 %	0.45 %

$\rho = .99$ convergencia a los valores de las medias es mucho más lento y pesar de haber hecho un **burn-in** de 10,000 muestras, todavía no está lo suficientemente de los valores reales como en el caso con $\rho = .8$.

Para subsecuentes experimentos podemos escoger el un valor de **burn-in** de 4,000 iteraciones para el caso $\rho = 0.8$ y uno de al menos 8,000 iteraciones para el caso $\rho = 0.99$, de lo que podemos observar en que momento (casi) converge.

A continuación presentamos algunas medidas del porcentage de rechazos de cada propuesta.

Notemos que el porcentaje de rechazo es muy bajo para ambas propuestas y que se reduce mucho más en el caso $\rho = 0.99$.

Problema 2

Consideré los tiempos de falla $t_1, ..., t_n$ con distribución Weibull(α, λ):

$$f(t_i|\alpha, \lambda) = \alpha\lambda t_i^{\alpha} \exp -t_i^{\alpha}\lambda$$

Se asumen como a priori $\alpha \sim \exp(c)$ y $\lambda|\alpha \sim Gama(\alpha, b)$, por lo tanto,

$$f(\alpha; \lambda) = f(\vec{t}|\alpha, \lambda) f(\alpha, \lambda)$$

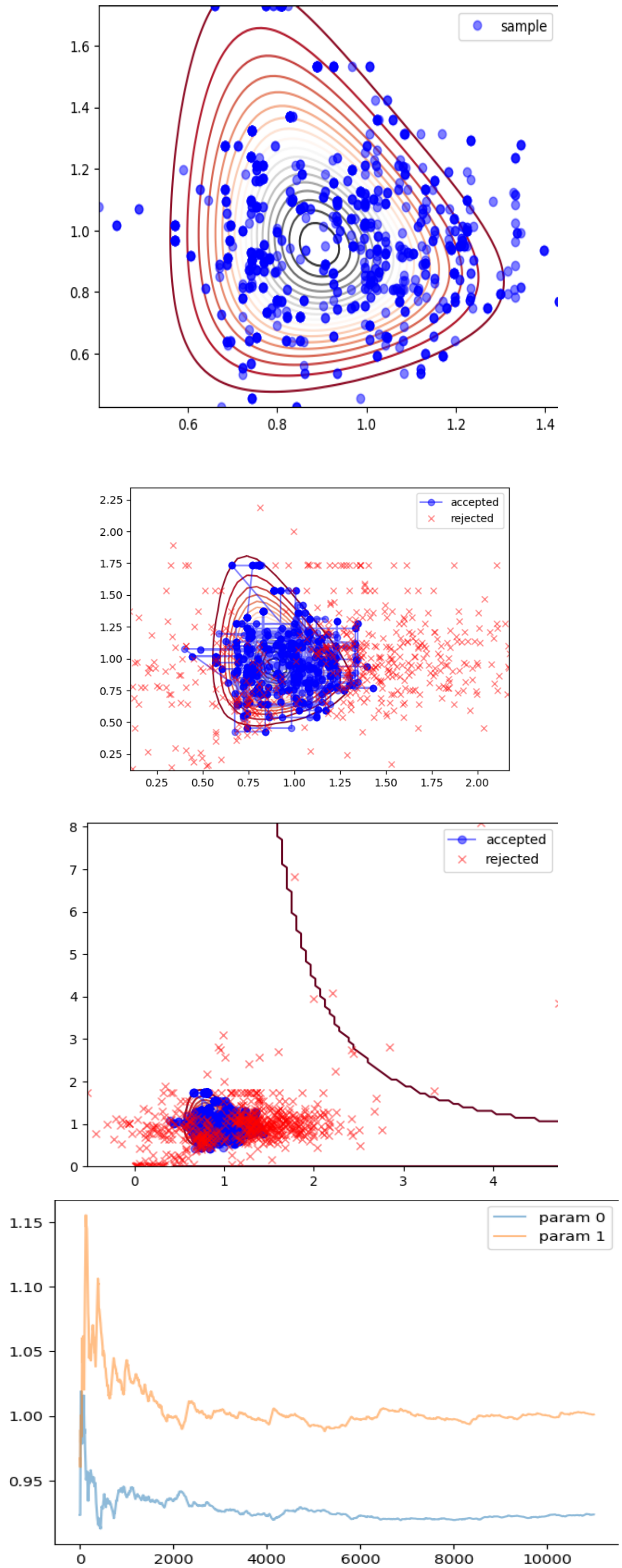
Así, para la distribución posterior se tiene:

$$f(\alpha, \lambda|\vec{t}, \vec{p}) \propto f(\vec{t}, \vec{p}|\alpha, \lambda) f(\alpha, \lambda)$$

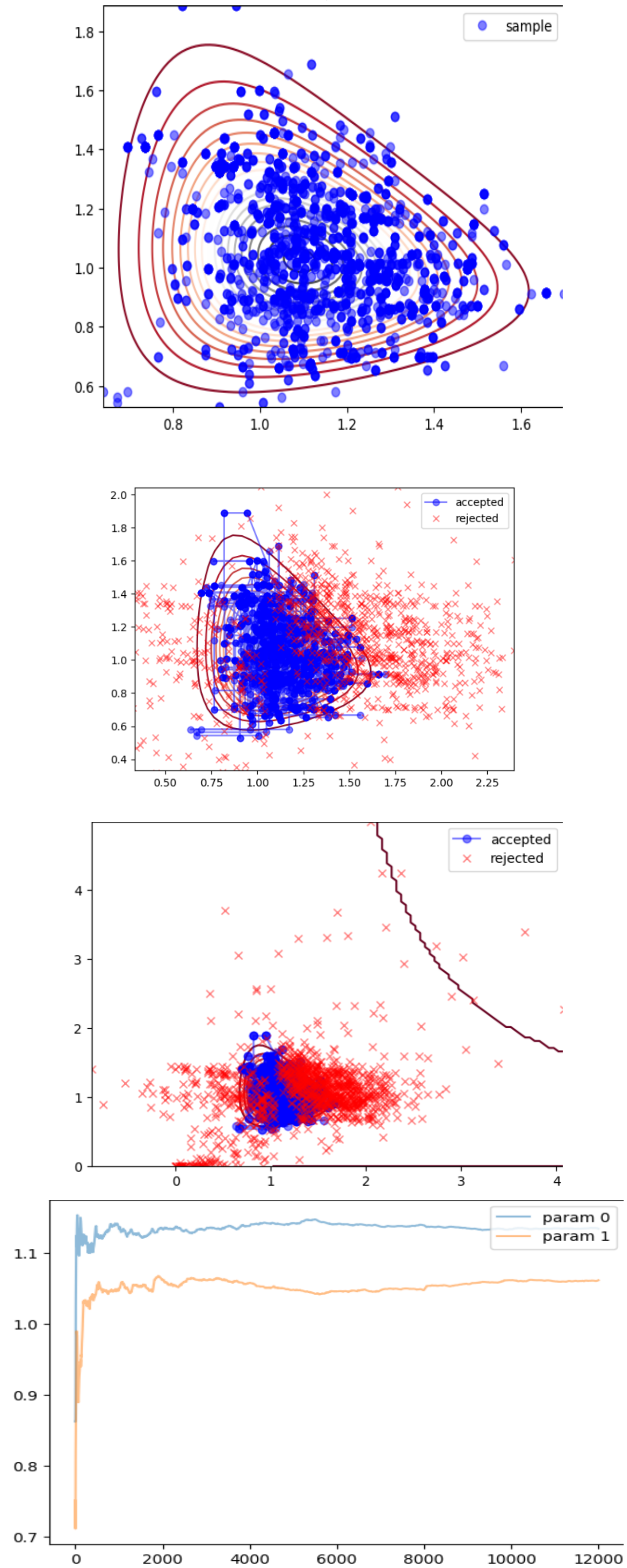
A partir del algoritmo MH usando Kerneles híbridos simule valores de la distribución posterior $f(\alpha, \lambda|\vec{t}, \vec{p})$.

Ejecutamos la implementación de nuestro con los siguientes parámetros. Escogimos un burn-in de 10,000 para estudiar la convergencia de los valores y muestras de tamaño $N = 1000, 2000$. Se utilizó una posición inicial con distribución $\mathcal{U}(0, 1)$ en cada entrada. Algo que es importante mencionar es que se utilizaron las siguientes probabilidades $p = .3, .3, .1, .3$ para utilizar cada uno de los 4 kerneles. Esto anterior para intentar reducir los rechazos en la propuesta 3, pues esta es la que más rechaza como veremos más adelante en la tabla de medida del rechazo.

N=1,000


sample
accepted
rejected
accepted
rejected
param 0
param 1

N=2,000


sample
accepted
rejected
accepted
rejected
param 0
param 1

De primera vista podemos observar que la muestra generada en ambos casos si tiene una cierta correspondencia con las curvas de nivel de la posterior. También podemos observar que hay muchos rechazos que ocurren muy lejos de la región con más densidad en la posterior. Una hipotesis de por que ocurre lo anterior, es que la curva de nivel transversal que observamos a la derechas puede estar atrayendo a la evolución del algoritmo. Notemos que en las últimas 2 gráficas las medias de la muestras generadas convergen alrededor de los valores reales $\alpha = \lambda = 1$.

A continuación observamos los porcentajes de rechazo para cada propuesta :

Parámetros	**Usos**					**# Rechazados**					**% Rechazados**				
	P1	**P2**	**P3**	**P4**	**Tot.**	**P1**	**P2**	**P3**	**P4**	**Tot.**	**P1**	**P2**	**P3**	**P4**	**Tot.**
Sample = 1,000	300	305	99	296	1000	11380	273	90	200	643	26.67 %	89.51 %	90.91 %	67.57 %	64.30 %
Sample = 2,000	616	620	176	588	2000	303	423	163	355	1244	49.19 %	68.61 %	92.61 %	60.37 %	62.20 %

Notemos que el porcentage de rechazo es muy alto en ambos casos. Esto no es gran sorpresa por que se puede corroborar esto mismo con las graficas donde se presentan propuestas aceptadas y rechazadas, donde veremos un gran número de rechazos muy lejos de región donde se concetra la densidad de la posterior.

Problema 3

Simule valores de la distribución posterior $f(\lambda_1, ..., \lambda_n | \vec{t}, \vec{p})$ usando kerneles híbridos.

Para verificar que las propuestas son de Gibbs, primero notemos que estamos utilizando en particular un **Random Scan Gibbs sampler**, puesto que en cada paso escogemos al azar una de las 11 variables $(\lambda_1, ..., \lambda_{10}, \beta)$ y la actualizamos con las siguientes propuestas.

$$\lambda_i | \beta, t_i, p_i \sim Gamma(p_i + \alpha, t_i + \beta), \quad (1 \leq i \leq 10),$$
$$\beta | \lambda_1, ..., \lambda_{10} \sim Gamma(\gamma + 10\alpha, \delta + \sum_{i=1}^{10} \lambda_i)$$

Lo primero que notamos es que realmente siguen la forma de propuestas Gibbs, puesto que se tiene una distribución condicional para cada parámetro λ_i, β dado todos los otros parámetros, e.g $\lambda_i | \lambda_{-i}, \beta$ o $\beta | \lambda_1, ..., , \lambda_{10}$. Támbien se verifica en las cuentas de la página 386 del libro **Monte Carlo Statistical Methods** [1] que realmente estas son las distribuciones condicionales obtenidas al desarrollar la función de densidad posterior con la prior's independientes $\lambda_i \sim Gamma(\alpha, \beta), \beta \sim Gamma(\gamma, \delta)$.

Utilizar las distribuciones condicionales mencionadas reales como propuestas es efectivamente lo que se hace con un **Gibb's sampler**, donde nos ahorramos la necesidad de construir propuestas de transición en el algoritmo de Metropolis-Hastings. Otra cosa que debemos notar es que una propuesta de Gibb's nunca rechazará una propuesta, osea que siempre $\rho = 1$.. Esto anterior los verificamos en nuestra implementación donde observamos un **0 %** de propuestas rechazadas, a comparación de los problemas anteriores.

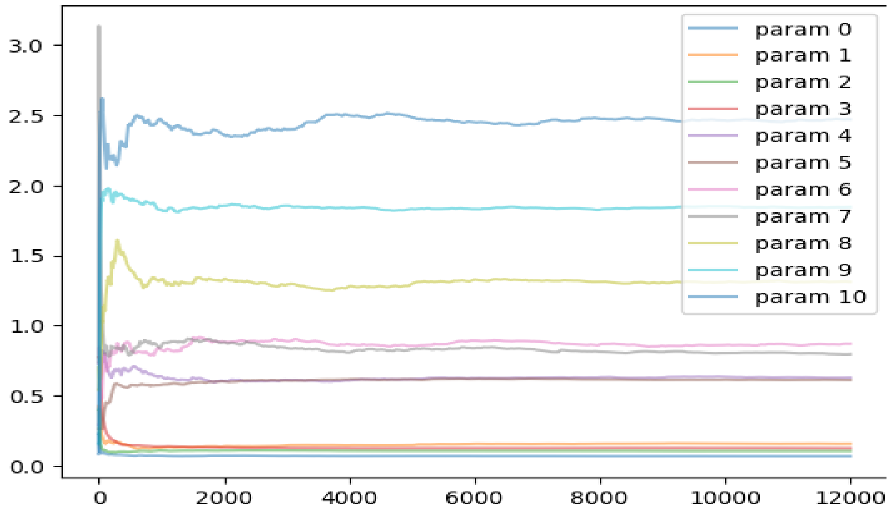
Utilizando los datos de **Fallos en bombas de agua en centrales nucleares** presentados en el enunciado y los parámetros para las distribuciones a priori, $\alpha = 1.8, \gamma = 0.01$ y $\delta = 1$, implementamos el algoritmo de Metropolis-Hastings con Kernels Híbridos para generar muestras de $(\lambda_1, ..., \lambda_{10}, \beta)$.

A continuación mostramos un exctracto de nuestra implementación de logaritmo de la posterior en python, basandonos completamente en la cuentras de la página 386 del libro **Monte Carlo Statistical Methods** [1].

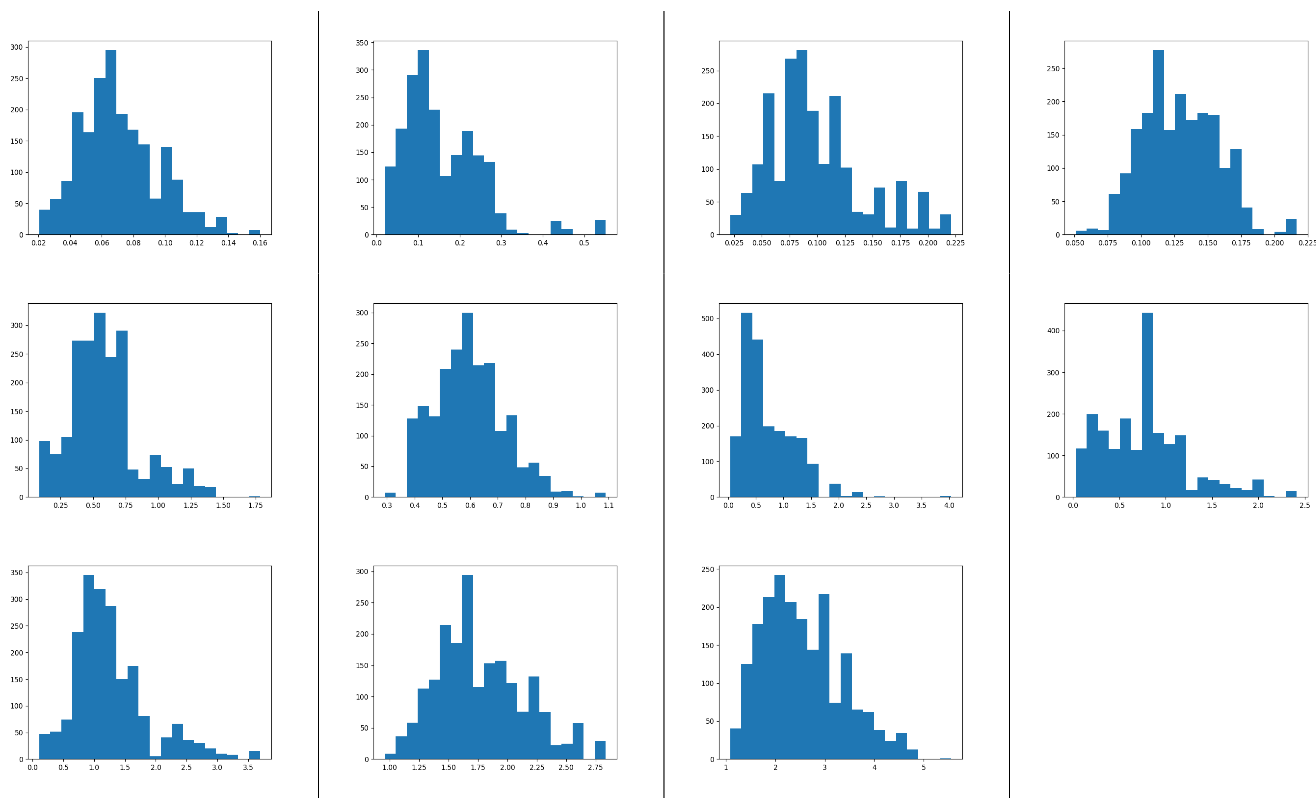
Algorithm 3: Posterior Normal

```
def log_posterior(x):
    lmbd_t, beta_t = x
    sm = 0

    # Verosimilitud de los datos
    for idx, lmbd_t_i in enumerate(lmbd_t):
        A = (p[idx] + alpha - 1) * np.log(lmbd_t_i)
        B = -(t[idx] + beta_t) * lmbd_t_i

        sm += A + B

    # Parte
    C = (10 * alpha + gamma - 1) * np.log(beta_t)
    D = -delta * beta_t

    return sm + C + D
```

A continuación presentamos las gráficas de las medias de las muestras generadas hasta cierta iteración t. Así que podemos estudiar la convergencia de las medias muestrales de cada $\lambda_1, ..., , \lambda_{10}, \beta$.



Claramente las medias estan convergiendo a cierto valor. Nuestro intento ahora será estudiar la posterior a traves de la muestras generadas por esa misma utilizando Metropolis-Hastings Otro estudio que podemos hacer es con histogramas en cada variable, estando en el orden $\lambda_1, ..., , \lambda_{10}, \beta$ de izquierda a derecha,



donde almenos podemos asegurar que cada variable está convergiendo a una distribución unimodal de lo que observamos en los histogramas. Sería muy interesante observar que tanto se acerca la media a una estimación burda $\hat{\lambda_i} = \frac{p_i}{t_i}$ utilizando los datos proporcionados. Intentaremos esto con una tabla simple comparando estas estimaciones.

	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9	λ_{10}	β
$\hat{\lambda_i} = \frac{p_i}{t_i}$	.05	.06	.07	.11	.57	.60	.95	.95	1.90	2.09	-
MCMC	.07	.15	.10	.12	.61	.60	.82	.84	1.28	1.85	2.44

Podemos notar que de alguna manera se acercan y preservan el orden nuestras estimaciones. Podríamos hacer un estudio más completo de la confiabilidad de la bombas de agua con estos resultados.

Referencias

[1] (Springer Texts in Statistics) - Monte Carlo Statistical Methods (2004), Christian Robert, George Casella