Metropolis-Hastings Algorithm

******* The Implemented code is inspired from this repository: [https://github.com/rodolformelo/Metropolis-Hastings-Algorithm/ (https://github.com/rodolformelo/Metropolis-Hastings-Algorithm/)](https://github.com/rodolformelo/Metropolis-Hastings-Algorithm/) and then modified with the aims of this project work.***

The Metropolis-Hastings algorithm is an example of MCMC sampling methods. This algorithm generates a set of random variables $\left\{\theta^{(t)}\right\}$ continuously for $t = 1, 2, \ldots,$ provided it has a joint posterior distribution $\pi(\theta \mid \mathbf{x})$. To produce these samples, this algorithm requires us to determine the proposed density $p\left(\theta^{(t+1)}, \theta^{(t)}\right)$. This function is the probability density function of θ at time t+1, given its value at time t. Therefore, the performance of the Metropolis-Hastings algorithm is expressed as follows

$$\theta^{(t+1)} \sim p\left(\theta^{(t+1)}, \theta^{(t)}\right)$$

$$\theta^{(t)} = \begin{cases} \theta^{(t+1)} & \rho\left(\theta^{(t)}, \theta^{(t+1)}\right) \\ \theta^{(t)} & 1 - \rho\left(\theta^{(t)}, \theta^{(t+1)}\right) \end{cases}$$

$$\rho\left(\theta^{(t)}, \theta^{(t+1)}\right) = \min\left\{1, \frac{f\left(\mathbf{x} \mid \theta^{(t+1)}\right)\pi\left(\theta^{(t+1)}\right)}{f\left(\mathbf{x} \mid \theta^{(t)}\right)\pi\left(\theta^{(t)}\right)} \frac{p\left(\theta^{(t)}, \theta^{(t+1)}\right)}{p\left(\theta^{(t+1)}, \theta^{(t)}\right)}\right\}$$

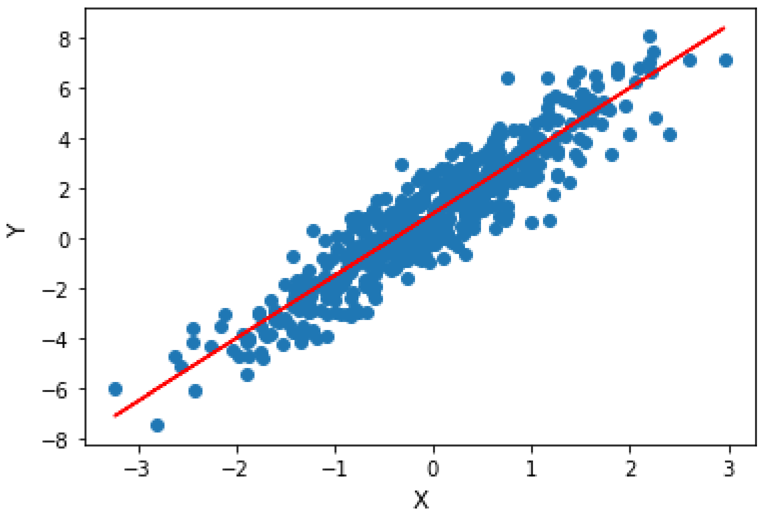
Task 1

- ## Univariate case

$$\mathrm{y} = \beta_0 + \beta_1 x + np.random.normal(loc = 0, scale = \mathrm{sigma}, size = (n, 1))$$

$$\mathrm{y} = 1 + 2.5x + np.random.normal(loc = 0, scale = \mathrm{sigma}, size = (n, 1))$$

True Values of β_0, β_1, and size of the samples :



Real Value:

β_0	1
β_1	2.5
σ	1
$size$	500

$$Estimation:$$

β_0	0.7336
β_1	1.8493
σ	3.893

Percentage absolute relative error (PARE):

β_0	29.16%
β_1	48.34%

- **Effect of using different proposals:**

Proposal Distribution	Normal	Beta	Uniform	Poisson with lambda = 0.5	Gamma
Acceptance Rate	**0.5592**	0.02026	1	0.01246	0.06458
$\boldsymbol{y}_{\text{mean}}$	**0. 8629**	0.8629	0.8629	0.8629	0.8629
$\boldsymbol{\beta_0}$	**0.7336**	0.8948	0.8822	0.9296	0. 5783
$\boldsymbol{\beta_1}$	**1.8493**	0.8257	0.5	1.725	1.3567

- **σ Estimation:**

$$\begin{aligned}
\mathrm{Var}(Y) &= \mathrm{Var}(\beta_0 + \beta_1 X + \epsilon) \\
&= \mathrm{Var}(\beta_1 X + \epsilon) \\
&= \mathrm{Var}(\beta_1 X) + \mathrm{Var}(\epsilon) \\
&= (\beta_1)^2\, \mathrm{Var}(X) + \mathrm{Var}(\epsilon)
\end{aligned}$$

In [257]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, multivariate_normal, bernoulli
from statsmodels.discrete.discrete_model import Probit
from statsmodels.tsa.stattools import acf
from statsmodels.graphics.tsaplots import plot_acf
np.random.seed(123)
```

In [258]:

```
np.random.seed(123)
n = 500 # number of observations
k = 1 # k = number of betas -1

X = np.ones((n,k+1))
X[:,1:] = np.random.normal(loc=0,scale=1,size=k*n).reshape(n,k)
beta_true = np.matrix( [1,2.5]).T
sigma_true = 1
eta = np.dot(X,beta_true) + np.random.normal(loc=0, scale = sigma_true, size=(n,1))
```

In [259]:

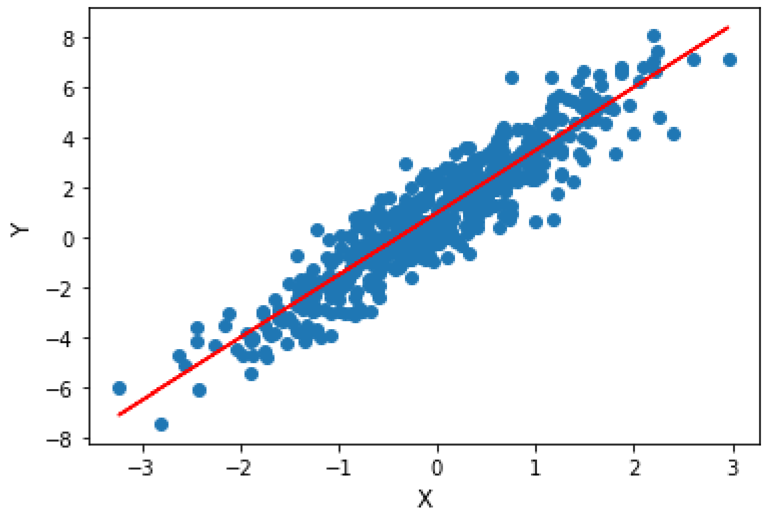
```
X0 = np.array(X[:,0].reshape(-1,1))
X1 = np.array(X[:,1].reshape(-1,1))
YY = np.array(eta)
```

In [260]:

```
#plt.scatter(X0,Y)
Y =  bernoulli.rvs(p = norm.cdf(eta))
YY.mean()
plt.scatter(X1,YY)
plt.plot(X1,X0+2.5*X1,color='red')
plt.xlabel('X', fontsize=12)
plt.ylabel('Y', fontsize=12)
Y =  bernoulli.rvs(p = norm.cdf(eta))
YY.mean()
```

Out[260]:

0.8629057675686005



In [261]:

```
def Li(beta, x, y): # likelihood calculations
    q = 2*y - 1
    eta = np.dot(x,beta)
    eta_q = np.multiply(q.T,eta)
    p = norm.cdf(eta_q)
    likehood = np.prod(p)
    return likehood

def Pr(beta, lambda0=10): #prior
    mean = np.zeros(shape=(beta.shape[0],))
    cov = (1/lambda0) * np.identity(beta.shape[0])
    prior_distribution = multivariate_normal(mean=mean, cov=cov)
    return prior_distribution.pdf(beta)

def POST(beta, x, y, lambda0=10): #posterire
    return Pr(beta, lambda0)*Li(beta, x, y)

#Work in log form for numeric stability
def ACCPT(beta_new, beta_old, x, y, lambda0=10):
    PN = POST(beta_new, x, y, lambda0)
    PO = POST(beta_old, x, y, lambda0)
    logratio = np.log(PN) - np.log(PO)
    ratio = np.exp(logratio)
    return min(1, ratio)

def COV(beta,x):
    np.seterr(divide = 'ignore')
    n,p = x.shape
    eta = np.dot(x,beta)
    numerator = norm.pdf(eta)**2
    denominator = (norm.cdf(eta))*(1.0001-norm.cdf(eta))
    value = np.divide(numerator, denominator)
    W = value * np.identity(n)
    I = x.T @ W @ x
    invI = np.linalg.inv(I)
    return invI

def MH(Y,X,lambda0 = 10, interations=5000, start=0, tau=1):
    n,p = X.shape
    np.random.seed(0)
    beta_tried = np.zeros(shape=(p,interations))
    beta_out = np.zeros(shape=(p,interations))
    beta_old = np.random.multivariate_normal(mean=start * np.ones((p,)), cov=np.ider
    acpt = 0
    for i in range(interations):
      covariance = COV(beta_old,X)
      beta_new = np.random.multivariate_normal(mean=beta_old, cov= tau * covariance)
      beta_tried[:,i] = beta_new.T
      u = np.random.rand()
      alpha = ACCPT(beta_new, beta_old, X, Y, lambda0)
      if u < alpha:
          beta_old = beta_new
          acpt+=1
      beta_out[:,i] = beta_old.T
    acpt_rate = acpt/interations
    return beta_out, beta_tried, acpt_rate
```

In [262]:

```
interations = 10000
beta, beta_tried, acpt_rate = MH(Y=Y,X=X, interations=interations, tau=1, lambda0=1,
print(f"The acceptance rate: {np.round(acpt_rate*100,4)}% \n")
for n in range(beta.T[0].shape[0]):
    true = float(beta_true[n])
    m = np.around(np.mean(beta[n,:]),4)
    stdev = np.around(np.std(beta[n,:]),4)
    print(f"Beta({n}) mean prediction: {m}, true_beta: {true} \n")
```

```
The acceptance rate: 55.92%

Beta(0) mean prediction: 0.7336, true_beta: 1.0

Beta(1) mean prediction: 1.8493, true_beta: 2.5
```

In [263]:

```
beta0 = np.around(np.mean(beta[0,:]),4)
beta1 = np.around(np.mean(beta[1,:]),4)
```

In [264]:

```
np.var(eta)-(beta0**2)*np.var(X0)-(beta1**2)*np.var(X1)
```

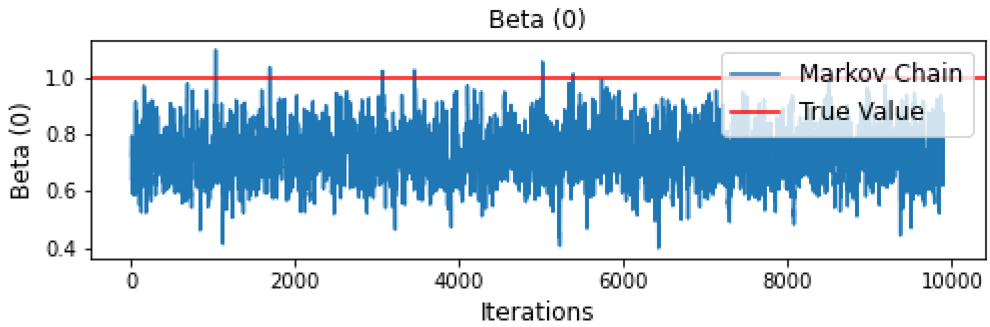
Out[264]:

```
3.8930003881077164
```

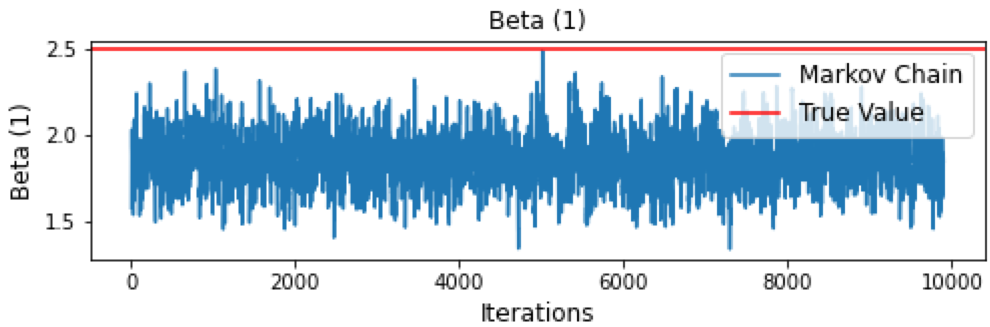
In [265]:

```
for n in range(beta.shape[0]):
    print(f"\n BETA({n}) \n")
    plt.figure(figsize=(8,2))
    plt.title(f'Beta ({n})')
    plt.plot(beta[n,100:].T)
    plt.axhline(beta_true[n], color='r', linestyle='-')
    plt.xlabel('Iterations', fontsize=12)
    plt.ylabel(f'Beta ({n})', fontsize=12)
    plt.legend(['Markov Chain','True Value'], fontsize=12, loc=1)
    #plt.ylim((0,1.25))
    plt.show()
```

```
 BETA(0)
```



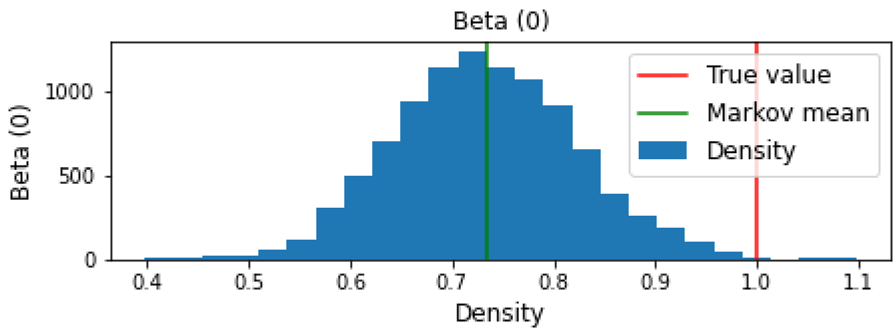

```
 BETA(1)
```



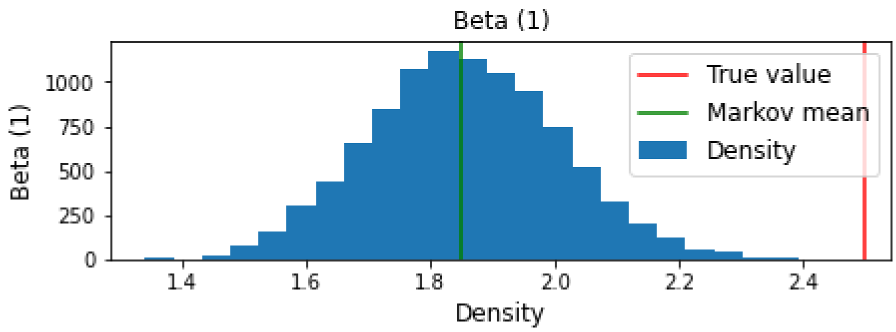
In [266]:

```
for n in range(beta.shape[0]):
    print(f"\n BETA({n}) \n")
    plt.figure(figsize=(7,2))
    plt.title(f'Beta ({n})')
    plt.hist(beta[n,100:], bins=25)
    plt.axvline(beta_true[n], color='r', linestyle='-')
    plt.axvline(np.mean(beta[n,:]), color='green', linestyle='-')
    #plt.axvline(MLE[n], color='blue', linestyle='-')
    plt.xlabel('Density', fontsize=12)
    plt.ylabel(f'Beta ({n})', fontsize=12)
    plt.legend(['True value', 'Markov mean','Density'], fontsize=12, loc=1)
    plt.show()
```

```
BETA(0)
```

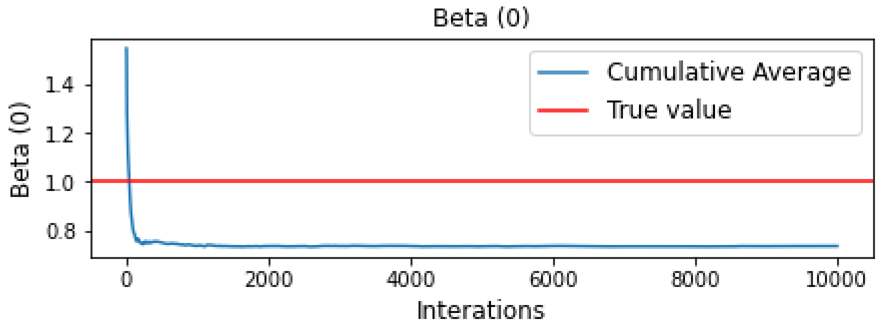


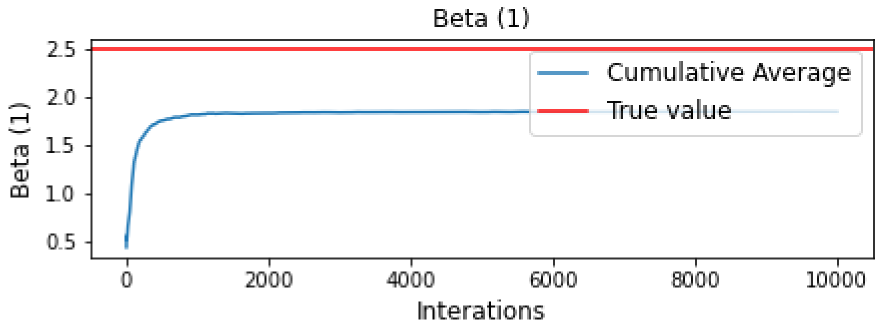

```
BETA(1)
```



In [267]:

```
for n in range(beta.shape[0]):
    beta_avg_cum = np.zeros(shape=(interations,))
    for t in range (interations):
        beta_avg_cum[t] = np.sum(beta[n,:t+1])/(t+1)
    plt.figure(figsize=(7,2))
    plt.title(f'Beta ({n})')
    plt.plot(beta_avg_cum.T)
    plt.axhline(beta_true[n], color='r', linestyle='-')
    plt.xlabel('Interations', fontsize=12)
    plt.ylabel(f'Beta ({n})', fontsize=12)
    #plt.ylim(0.25,2.5)
    plt.legend(['Cumulative Average','True value'], fontsize=12, loc=1)
    plt.show()
```





In [ ]:

```

```

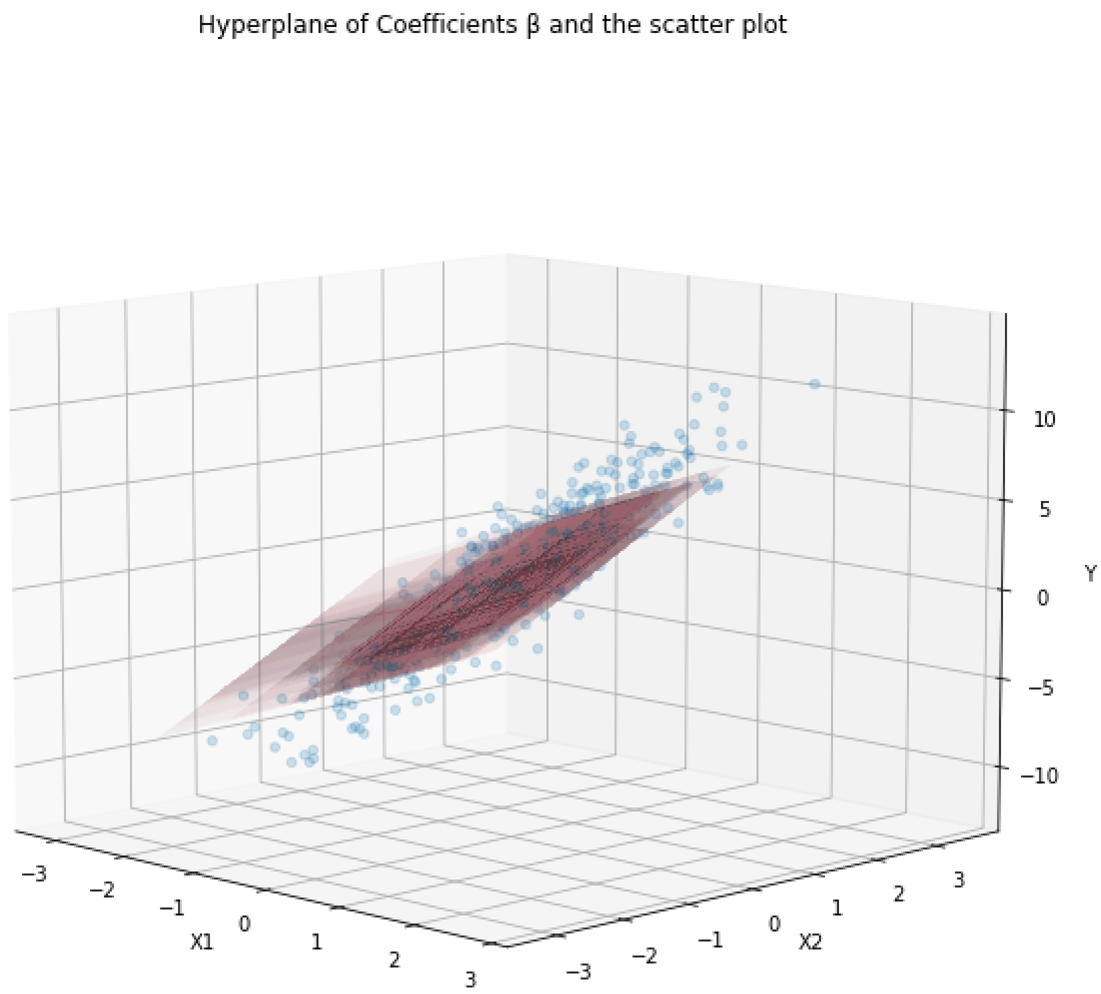
Task 2

- **Bivariate case**

$$y = \beta_0 + \beta_1 x1 + \beta_2 x2 + np.random.normal(loc = 0, scale = \text{sigma}, size = (n, 1))$$

$$y = 1 + 2.5x1 + 2.8x2 + np.random.normal(loc = 0, scale = \text{sigma}, size = (n, 1))$$

True Values of β_0, β_1, β_2 and size of the samples :



Real Valuse:

β_0	1
β_1	2.5
β_2	2.8
σ	1
$size$	500

$Estimation$:

β_0	0.8433
β_1	1.8179
β_2	2.1354
σ	7.1888

Percentage absolute relative error (PARE):

β_0	59.8866%
β_1	36.0453%
β_2	42.8976%

In [271]:

```
np.random.seed(123)
n = 500
k = 2

X = np.ones((n,k+1))
X[:,1:] = np.random.normal(loc=0,scale=1,size=k*n).reshape(n,k)
beta_true = np.matrix( [1,2.5,2.8]).T
sigma_true = 1
eta = np.dot(X,beta_true) + np.random.normal(loc=0, scale = sigma_true, size=(n,1))

Y =  bernoulli.rvs(p = norm.cdf(eta))
YY.mean()
interations = 10000
beta, beta_tried, acpt_rate = MH(Y=Y,X=X, interations=interations, tau=1, lambda0=1,
print(f"The acceptance rate: {np.round(acpt_rate*100,4)}% \n")
for n in range(beta.T[0].shape[0]):
    true = float(beta_true[n])
    m = np.around(np.mean(beta[n,:]),4)
    stdev = np.around(np.std(beta[n,:]),4)
    print(f"Beta({n}) mean prediction: {m}, true_beta: {true} \n")
```

```
The acceptance rate: 45.38%

Beta(0) mean prediction: 0.8433, true_beta: 1.0

Beta(1) mean prediction: 1.8179, true_beta: 2.5

Beta(2) mean prediction: 2.1354, true_beta: 2.8
```

In [272]:

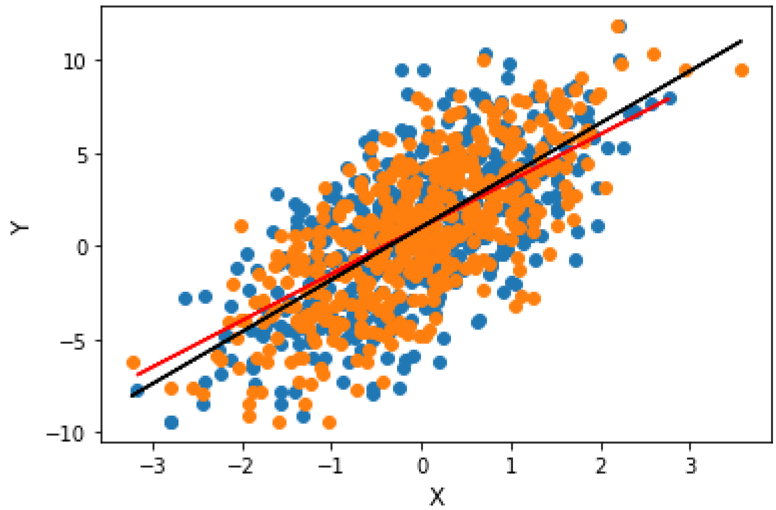
```
X0 = np.array(X[:,0].reshape(-1,1))
X1 = np.array(X[:,1].reshape(-1,1))
X2 = np.array(X[:,2].reshape(-1,1))
YY = np.array(eta)
```

In [273]:

```
#plt.scatter(X0,Y)
plt.scatter(X1,YY)
plt.scatter(X2,YY)
plt.plot(X1,X0+2.5*X1,color='red')
plt.plot(X2,X0+2.8*X2,color='black')
plt.xlabel('X', fontsize=12)
plt.ylabel('Y', fontsize=12)
```

Out[273]:

```
Text(0, 0.5, 'Y')
```



In [274]:

```
beta0 = np.around(np.mean(beta[0,:]),4)
beta1 = np.around(np.mean(beta[1,:]),4)
beta2 = np.around(np.mean(beta[2,:]),4)
```

In [275]:

```
np.var(eta)-(beta0**2)*np.var(X0)-(beta1**2)*np.var(X1)-(beta2**2)*np.var(X2)
```

Out[275]:

```
7.188810125992467
```

In [293]:

```
## PARE Calculations:
beta_means = (beta0+beta1+beta2)/3
PARE = abs((beta_means-beta_true)/beta_true)*100
PARE
```

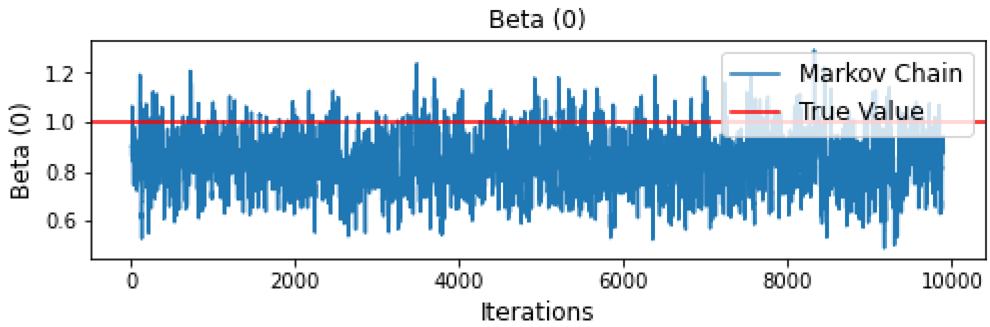
Out[293]:

```
matrix([[59.88666667],
        [36.04533333],
        [42.89761905]])
```

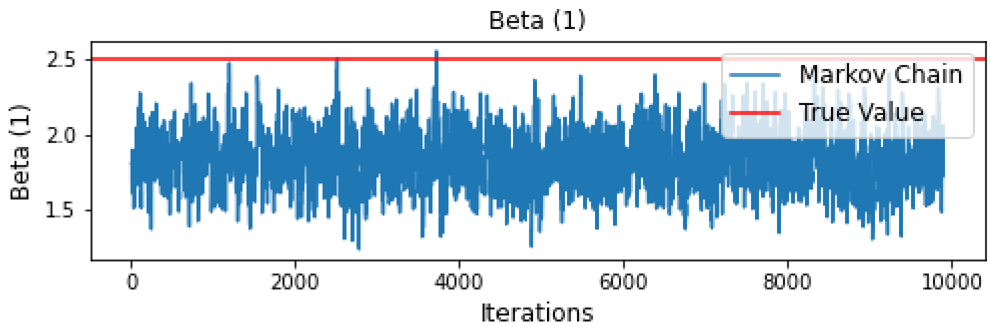
In [276]:

```
for n in range(beta.shape[0]):
    print(f"\n BETA({n}) \n")
    plt.figure(figsize=(8,2))
    plt.title(f'Beta ({n})')
    plt.plot(beta[n,100:].T)
    plt.axhline(beta_true[n], color='r', linestyle='-')
    plt.xlabel('Iterations', fontsize=12)
    plt.ylabel(f'Beta ({n})', fontsize=12)
    plt.legend(['Markov Chain','True Value'], fontsize=12, loc=1)
    #plt.ylim((0,1.25))
    plt.show()
```

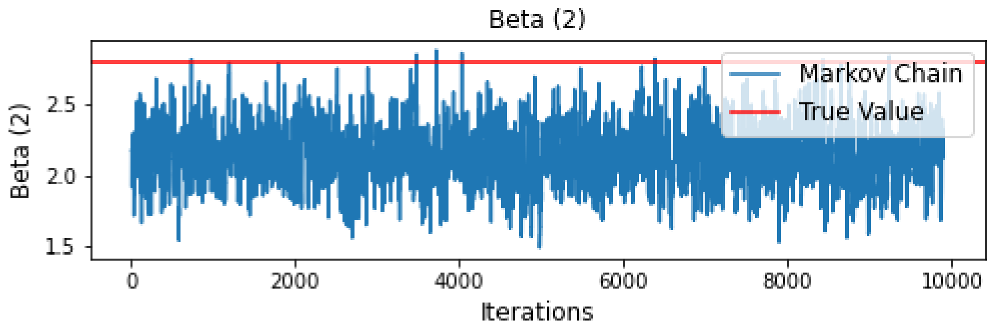
```
 BETA(0)
```




```
 BETA(1)
```



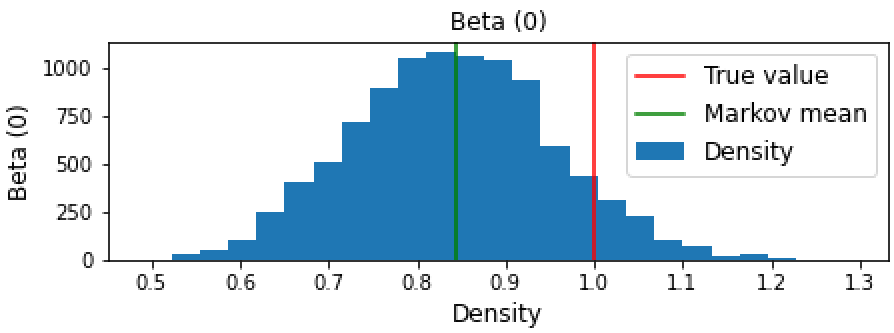

```
 BETA(2)
```



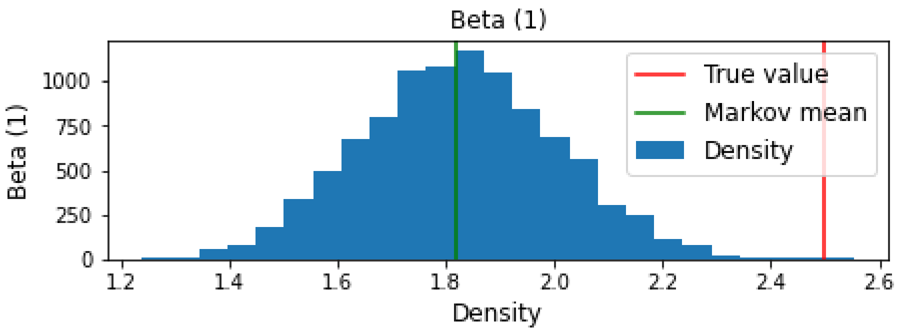
In [277]:

```
### Histogram
for n in range(beta.shape[0]):
    print(f"\n BETA({n}) \n")
    plt.figure(figsize=(7,2))
    plt.title(f'Beta ({n})')
    plt.hist(beta[n,100:], bins=25)
    plt.axvline(beta_true[n], color='r', linestyle='-')
    plt.axvline(np.mean(beta[n,:]), color='green', linestyle='-')
    #plt.axvline(MLE[n], color='blue', linestyle='-')
    plt.xlabel('Density', fontsize=12)
    plt.ylabel(f'Beta ({n})', fontsize=12)
    plt.legend(['True value', 'Markov mean','Density'], fontsize=12, loc=1)
    plt.show()
```

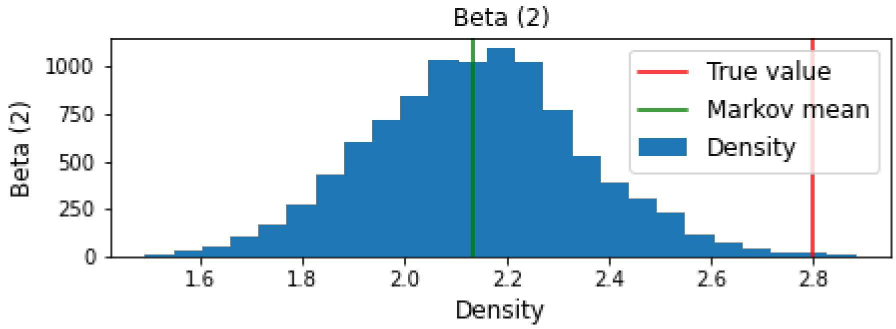
```
BETA(0)
```




```
BETA(1)
```

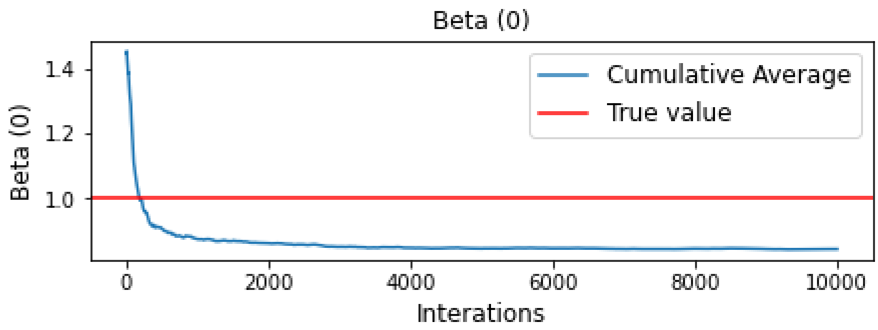


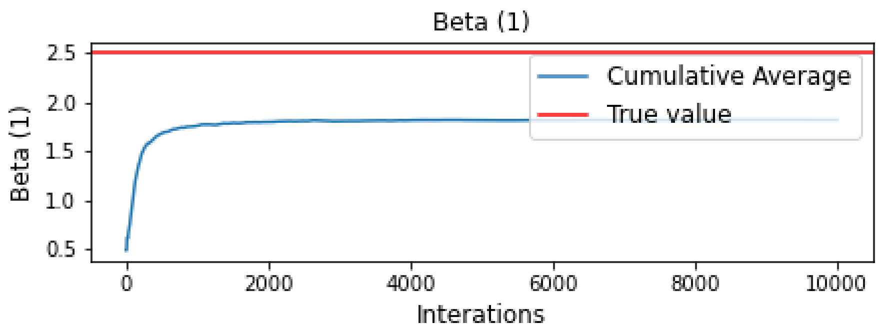

```
BETA(2)
```

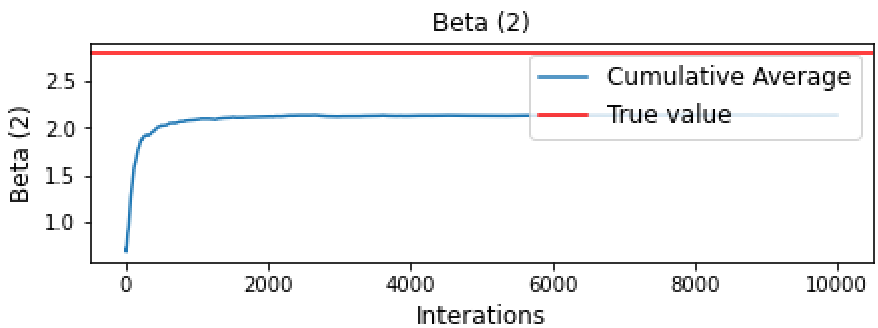


In [254]:

```
for n in range(beta.shape[0]):
    beta_avg_cum = np.zeros(shape=(interations,))
    for t in range (interations):
        beta_avg_cum[t] = np.sum(beta[n,:t+1])/(t+1)
    plt.figure(figsize=(7,2))
    plt.title(f'Beta ({n})')
    plt.plot(beta_avg_cum.T)
    plt.axhline(beta_true[n], color='r', linestyle='-')
    plt.xlabel('Interations', fontsize=12)
    plt.ylabel(f'Beta ({n})', fontsize=12)
    #plt.ylim(0.25,2.5)
    plt.legend(['Cumulative Average','True value'], fontsize=12, loc=1)
    plt.show()
```







In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:

```
np.random.seed(123)
n = 500 # number of observations
k = 2 # k = number of betas -1

X = np.ones((n,k+1))
X[:,1:] = np.random.normal(loc=0,scale=1,size=k*n).reshape(n,k)
beta_true = np.matrix( [1,2.5,2.8]).T
sigma_true = 1
eta = np.dot(X,beta_true) + np.random.normal(loc=0, scale = sigma_true, size=(n,1))
```

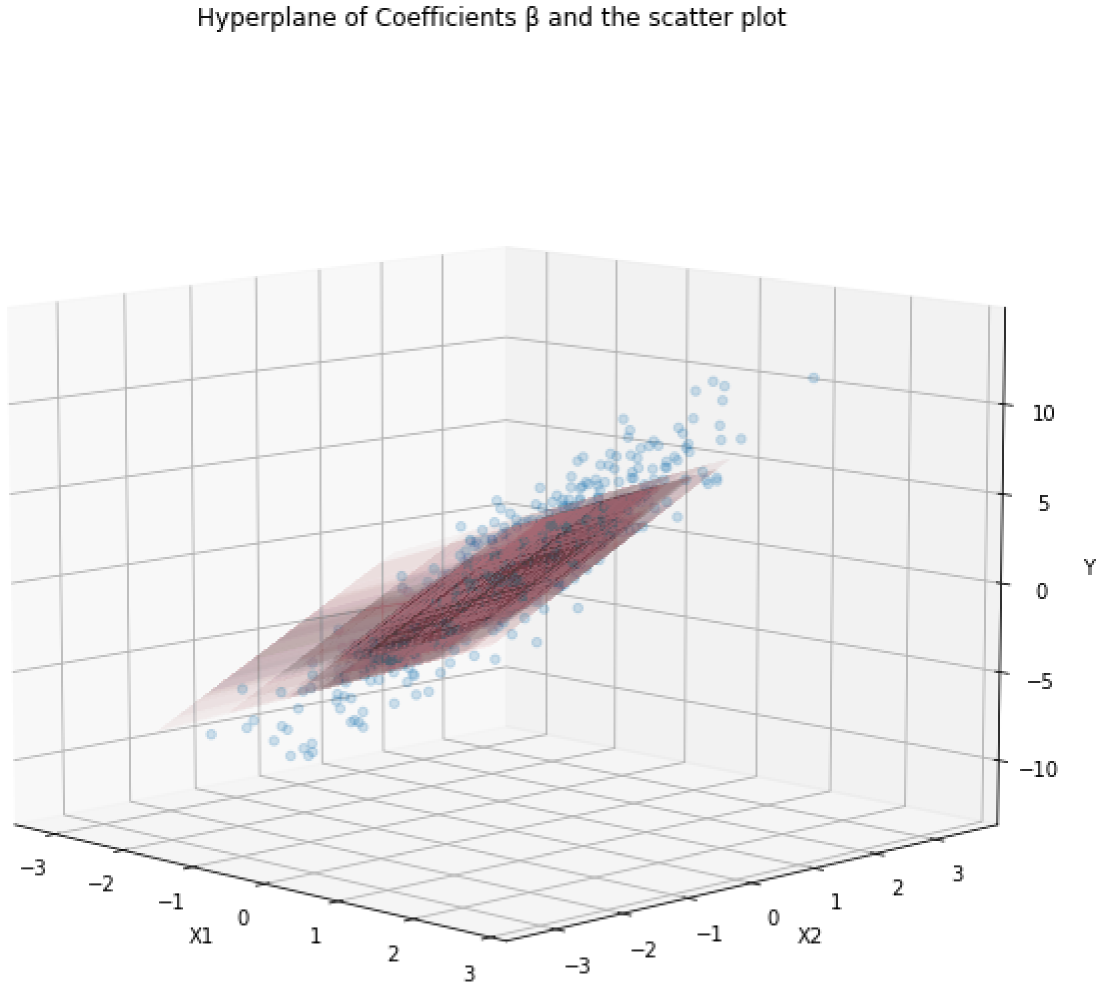
In [4]:

```
X0 = np.array(X[:,0].reshape(-1,1))
X1 = np.array(X[:,1].reshape(-1,1))
X2 = np.array(X[:,2].reshape(-1,1))
YY = np.array(eta)
```

In [11]:

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
fig.set_size_inches(15, 10)
ax = fig.add_subplot(111, projection='3d')
ax.scatter(xs=X1, ys=X2, zs=YY,alpha=0.2)

XX, XY = np.meshgrid(X1, X2)
eq2= 1+2.5*XX+2.8*XY
eq = 0.8433+1.8179*XX+2.1354*XY
ax.plot_surface(XX, XY, eq,alpha=0.01, color="pink")
#ax.plot_surface(XX, XY, eq2,alpha=0.01, color="black")
ax.set_ylabel('X2'); ax.set_xlabel('X1'); ax.set_zlabel('Y')
ax.view_init(10, -45)
plt.title('Hyperplane of Coefficients β and the scatter plot')
plt.show()
```



In [ ]:

In [ ]: