gibbs

July 11, 2019

```
In [1]: import numpy as np
        import pandas as pd
        from scipy import stats
        import matplotlib.pyplot as plt

        from scipy.special import digamma
```

0.1 Make the function

Make the required functions, which calculate XX', $expit$ and draw the autocorelation plot

```
In [2]: def bernoulliSample(p=0.5,n=1):
            lst = []
            for i in range(n):
                if p > np.random.uniform(low=0,high=1):
                    lst.append(1)
                else:
                    lst.append(0)
            if n==1:
                return(lst[0])
            else:
                return(np.array(lst))
```

```
In [3]: def product(a):
            n = len(a)
            out = np.zeros([n,n])
            for i in range(n):
                for j in range(n):
                    out[i,j] = a[i]*a[j]
            return(out)
```

```
In [4]: def expit(x):
            #if x < 100:
            return(np.exp(x)/(1+np.exp(x)))
            #else:
                #return(1)
```

```
In [5]: def acf(sampl,lag =40):
            sampl= np.array(sampl)
            base = sampl.dot(sampl)/len(sampl)
            acr = [1]
            for t in range(1,lag):
                acr.append((sampl[t:].dot(sampl[:-t])/(len(sampl)-t))/base)
            x = list(range(lag))
            plt.bar(x,acr,color='gray')
            plt.title('Autocorrelation plot')
            plt.hlines(0.05,xmin=-0.5 ,xmax=lag,colors='r',linestyles='dashed')
            plt.hlines(-0.05,xmin=-0.5 ,xmax=lag,colors='r',linestyles='dashed')
            #plt.show()
```

```
In [90]: def hist(sampled,tlt,nbin =19,gr=4):
             grid =np.linspace(-gr,gr,1000)
             plt.hist(sampled, bins=nbin,color='gray',density=True)
             #plt.plot(grid , stats.norm.pdf(grid), 'black')
             plt.title(tlt)
             plt.xlabel('x')
             #plt.xlim(-4,4)
             plt.ylabel('density')
             #plt.show()
```

1 Gibbs Sampling

1.1 Prior

$$
\begin{aligned}
Y|\beta,\sigma^2,\Gamma &\sim N(Z\Gamma\beta,\sigma^2\cdot I)\\
\sigma^2 &\sim Inverse-Gamma(a,b)\\
\beta_j|\sigma^2_{\beta_j} &\sim^{ind} N(0,\sigma^2_{\beta_j})\\
\sigma^2_{\beta_j} &\sim^{iid} Inverse-Gamma(c,d)\\
\gamma_j &\sim^{iid} Bernoulli(\rho)\\
\rho &\sim Beta(u,v)
\end{aligned}
$$

where
Z is n x p design matrix
$\Gamma = diag(\gamma_j)$ for $j = 1,\dots,p$
a,b,c,d,u,v is flat prior

1.2 Posterior

$$
\begin{aligned}
p(\beta,\sigma^2,\sigma^2_\beta,\Gamma|Y) \propto & p(\beta,\sigma^2,\sigma^2_\beta,\Gamma,Y) \\
\propto & p(Y|\beta,\sigma^2,\Gamma)p(\beta|\sigma^2_\beta)p(\Gamma)p(\sigma^2)p(\sigma^2_\beta)p(\rho) \\
\propto & \left(\sigma^2\right)^{-n/2}\exp\left(-\frac{1}{2\sigma^2}\left(Y-Z\Gamma\beta\right)'\left(Y-Z\Gamma\beta\right)\right) \\
& \times\prod_{j=1}^{p}\left(\sigma^2_{\beta_j}\right)^{-1/2}\exp\left(-\frac{1}{2}\sum_{j=1}^{p}\frac{\beta_j^2}{\sigma^2_{\beta_j}}\right) \\
& \times\prod_{j=1}^{p}\rho^{\gamma_j}\left(1-\rho\right)^{1-\gamma_j} \\
& \times\left(\sigma^2\right)^{-a-1}\exp\left(-\frac{b}{\sigma^2}\right) \\
& \times\prod_{j=1}^{p}\left(\sigma^2_{\beta_j}\right)^{-c-1}\exp\left(-\sum_{j=1}^{p}\frac{d}{\sigma^2_{\beta_j}}\right) \\
& \times\rho^{u-1}\left(1-\rho\right)^{v-1}
\end{aligned}
$$

1.3 Sampling the β from

$$N\left(\mu,\Sigma\right)$$

where

$$\Sigma=\left(diag(\sigma^2_{\beta_j})+\frac{1}{\sigma^2}\Gamma'Z'Z\Gamma\right)^{-1},\quad \mu=\frac{1}{\sigma^2}\Sigma\Gamma'Z'y$$

```
In [7]: def sampleBeta(gamma,s2,sb2):
            Gamma = np.diag(gamma)
            D = np.diag(1/sb2)
            sinv = D + (1/s2)*Gamma.T.dot(Z.T.dot(Z.dot(Gamma)))
            Sigma = np.linalg.inv(sinv)
            mu = (1/s2)*Sigma.dot(Gamma.T).dot(Z.T).dot(y)
            out = np.random.multivariate_normal(mu,Sigma)
            return(out)
```

1.4 Sampling σ^2 from

$$Inverse-Gamma\left(a+\frac{N}{2},b+\frac{1}{2}\left(y-\Gamma\beta\right)'\left(y-\Gamma\beta\right)\right)$$

```
In [8]: def sampleS2(beta,gamma):
            Gamma = np.diag(gamma)
            alpha = a + N/2
            igbeta = b + 0.5*(y-Z.dot(Gamma).dot(beta)).T.dot((y-Z.dot(Gamma).dot(beta)))
            out = stats.invgamma.rvs(a=alpha,scale=igbeta)
            return(out)
```

1.5 Sampling $\sigma^2_{\beta_j}$ from

$$Inverse-Gamma\left(c+\frac{1}{2},d+\frac{1}{2}\beta_j^2\right)$$

```
In [9]: def sampleSb2(beta):
            lst = []
            for j in range(p):
                alpha = c + 1/2
                igbeta = d+0.5*(beta[j]**2)
                lst.append(stats.invgamma.rvs(a=alpha,scale=igbeta))
            out = np.array(lst)
            return(out)
```

1.6 Sampling γ_j from

$$Bernoulli\left(expit\left(logit(\rho)+\frac{\beta_j}{\sigma^2}Z_j'\left(y-Z_{-j}\Gamma_{-j}\beta_{-j}\right)-\frac{\beta_j^2}{2\sigma^2}Z_j'Z_j\right)\right)$$

$expit$ is inverse function of $logit$

```
In [10]: def samplegamma(rho,s2,beta,gamma):
             gam = gamma.copy()
             wlst = []
             for j in range(p):
                 Gamma= np.diag(gam)
                 eta = np.log(rho/(1-rho))-((beta[j]**2)/s2)*Z[:,j].T.dot(Z[:,j])\
                 +(beta[j]/s2)*Z[:,j].T.dot(y-np.delete(Z,j,1).dot(np.diag(np.delete(gamma,j)).d
                 w = expit(eta)
                 wlst.append(w)
                 gam[j] = w
             outlst = []
             for k in gam:
                 outlst.append(stats.bernoulli.rvs(k))
             out = np.array(outlst)
             return(out)
```

1.7 Sampling ρ from

$$Beta\left(\sum_{j=1}^{p}\gamma_j+u,p-\sum_{j=1}^{p}\gamma_j+v\right)$$

```
In [11]: def sampleRho(gamma):
             alpha = sum(gamma) +u
             bbeta = p-sum(gamma) + v
             out = np.random.beta(alpha,bbeta)
             return(out)
```

1.8 Gibbs sampling function

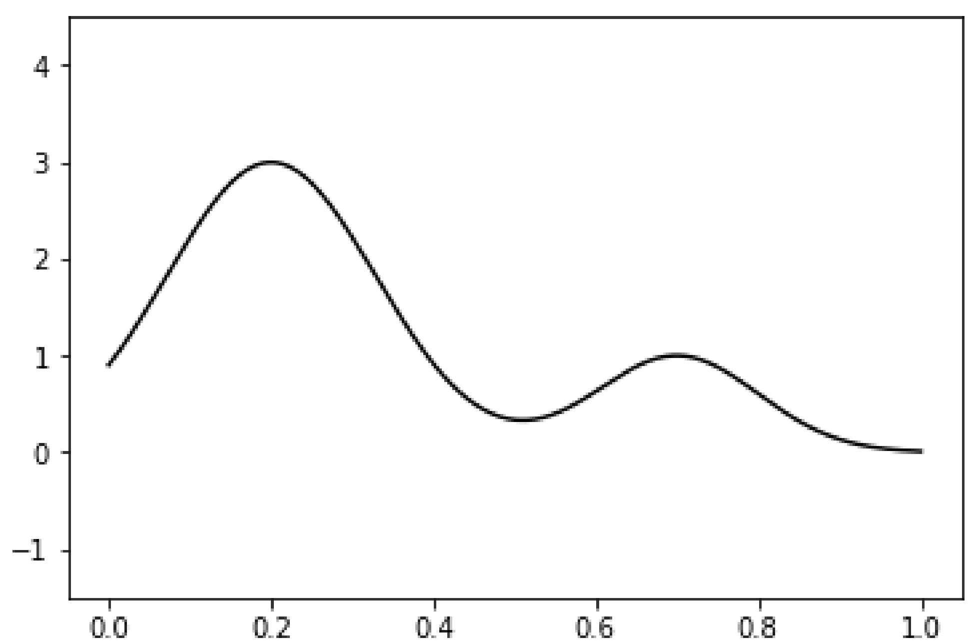
```
In [12]: def gibbs():
             lst = []
             for i in range(1000):
                 beta = sampleBeta(gamma,s2,sb2)
                 s2 = sampleS2(beta,gamma)
                 sb2 = sampleSb2(beta)
                 gamma = samplegamma(rho,s2,beta,gamma)
                 rho = sampleRho(gamma)

                 params = np.array([beta,s2,sb2,gamma,rho])
                 lst.append(params)
             return(pd.DataFrame(lst))
```

1.9 Simulation Data

$$f(x) = 3\exp(-30(x-0.2)^2) + \exp(-50(x-0.7)^2)$$

```
In [13]: def f(x):
             out = 3*np.exp(-30*((x-0.2)**2))+np.exp(-50*((x-0.7)**2))
             return(out)
```

```
In [14]: x = np.linspace(0,1,300)
         y = f(x)
         plt.plot(x, y, 'k')
         plt.ylim(-1.5, 4.5)
         plt.show()
```

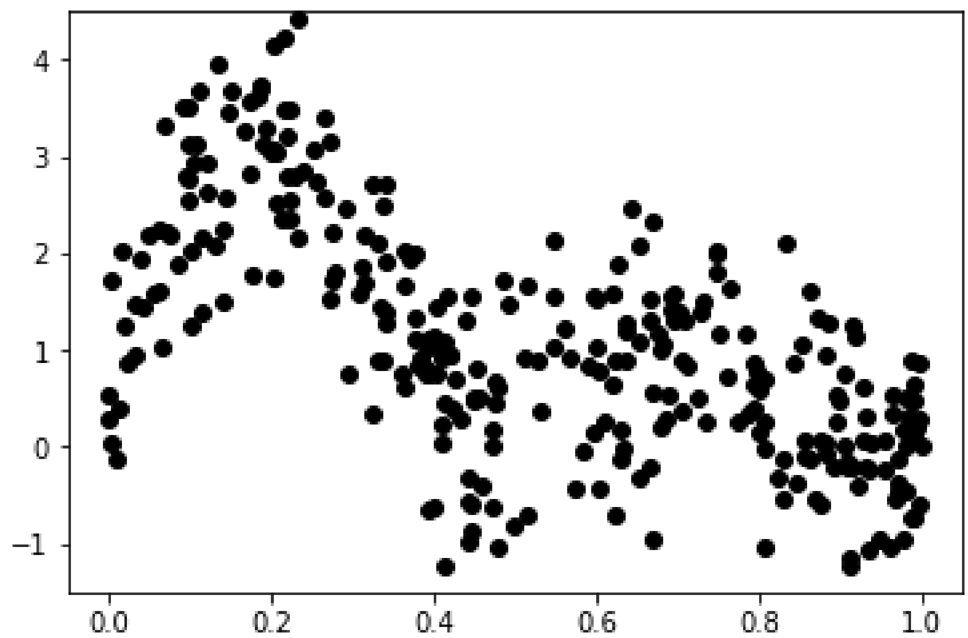


1.10 Make data wiht N(0,0.5) error

```
In [15]: def mkToy(n=300,tau = 0.5):
             x = np.random.uniform(size = n)
             e = np.random.normal(0,np.sqrt(0.5), size= n)
             y = f(x)+e
             #out = np.column_stack([x,y])
             return(x,y)
```

```
In [16]: x,y = mkToy()
```

```
In [17]: plt.plot(x,y,'ko')
         plt.ylim(-1.5, 4.5)
         plt.show()
```



1.11 Radial basis

we use radial basis functions defined by

$$\mathbf{b}(u) = \left\{ u, \left| \frac{u - \tau_1}{c} \right|^3, \cdots, \left| \frac{u - \tau_K}{c} \right|^3 \right\}$$

where c is sample standard deviation

```
In [18]: def defineKnot(X,K=10):
             upper = max(X)
```

```python
        lower = min(X)
        out = np.linspace(start=lower,stop=upper,num=K+2)[1:K+1]
        return(out)
    def radialbasis(u,tau,sd):
        lst = []
        lst.append(u)
        for i in tau:
            lst.append(abs((u-i)/sd)**3)
        out = np.array(lst)
        return(out)
```

In [19]:
```python
sd = np.std(x)
```

In [20]:
```python
knot = defineKnot(x)
d_x = radialbasis(x,knot,sd).T
```

In [86]:
```python
#initial value
Z = d_x
N, p = Z.shape
a,b,c,d = [10**-7]*4
e,f = [1,1]
gamma = bernoulliSample(0.5,p)
sb2 = np.repeat(0.5,p)
s2 = 0.5
rho=0.5
u,v = 1,1
print(gamma)
```

```
[0 1 1 1 0 0 0 0 1 0 0]
```

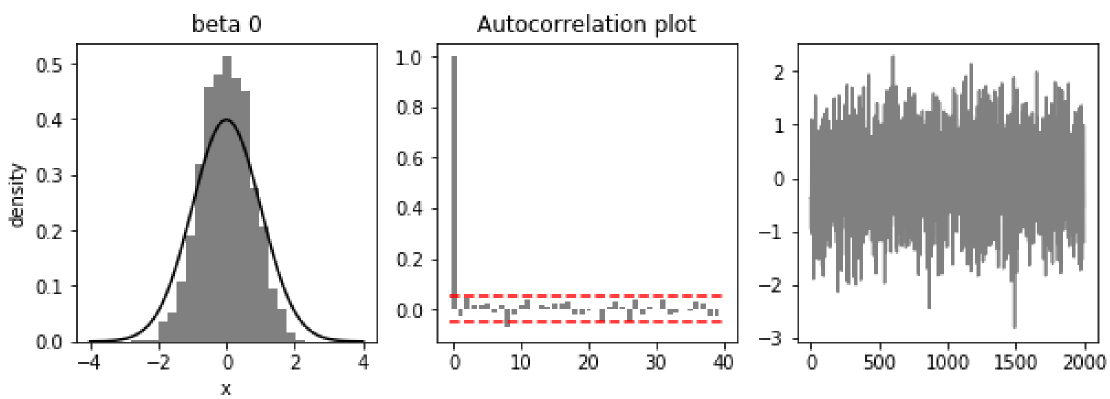
In [87]:
```python
lst = []
blst = []
rholst = []
s2lst = []
for i in range(2000):
    beta = sampleBeta(gamma,s2,sb2)
    s2 = sampleS2(beta,gamma);s2lst.append(s2)
    #sb2 = sampleSb2(beta)
    gamma = samplegamma(rho,s2,beta,gamma)
    rho = sampleRho(gamma);rholst.append(rho)

    #params = np.array([beta,s2,sb2,gamma,rho])
    #lst.append(params)
    blst.append(beta)
```

In [80]:
```python
df = pd.DataFrame(lst)
```

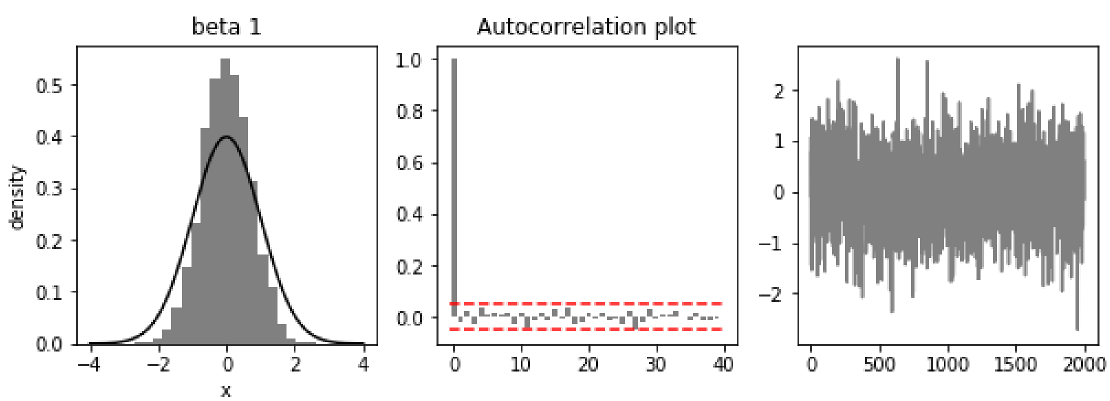
In [81]:
```python
bdf = pd.DataFrame(blst)
```

In [82]:
```
def betaplt(j):
    plt.figure(figsize=(10, 3))
    plt.subplot(1,3,1)
    hist(bdf[j],tlt='beta %d'%j)
    plt.subplot(1,3,2)
    acf(bdf[j])
    plt.subplot(1,3,3)
    plt.plot(bdf[j],color='gray')
    plt.show()
```

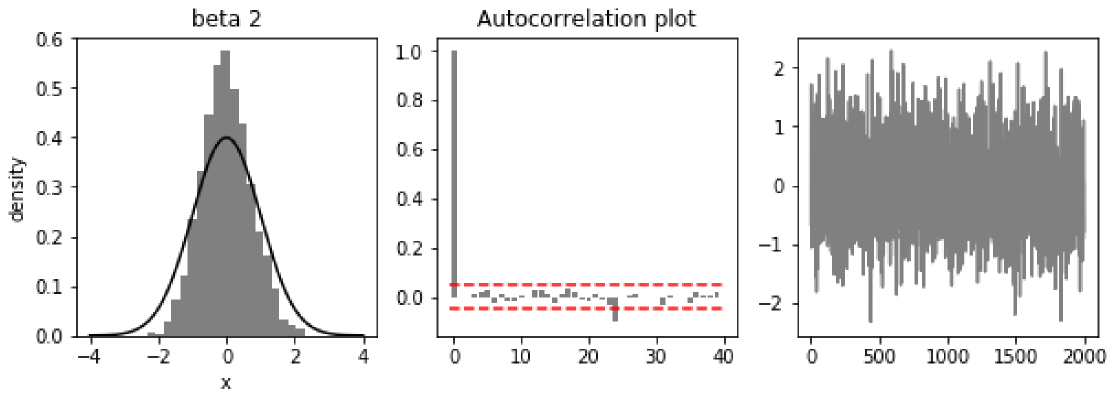
In [66]:
```
betaplt(0)
```



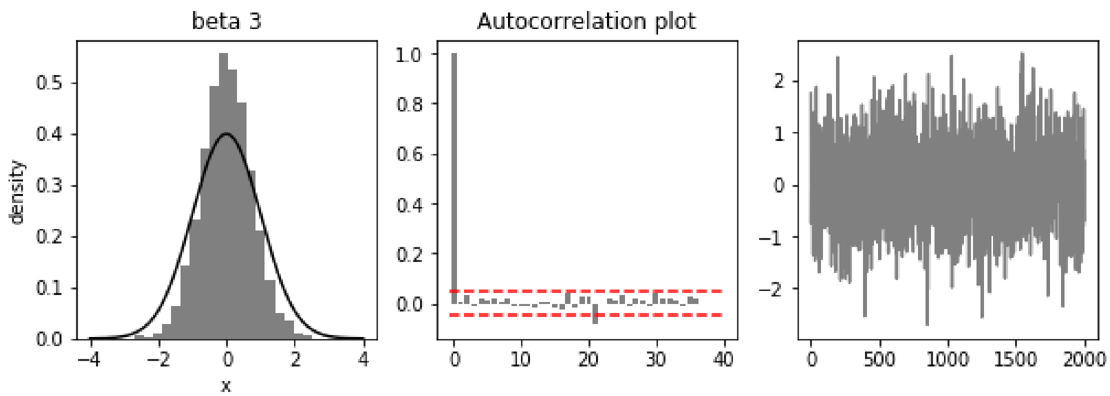
In [67]:
```
betaplt(1)
```



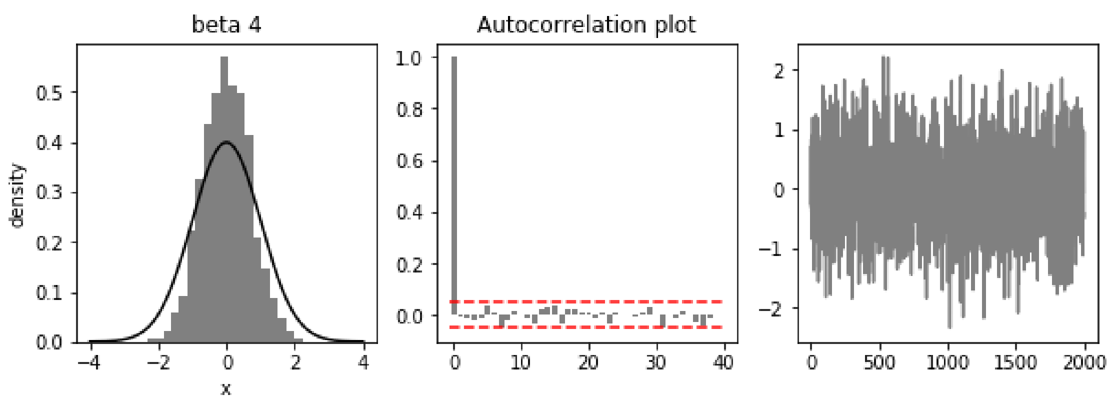
In [68]:
```
betaplt(2)
```



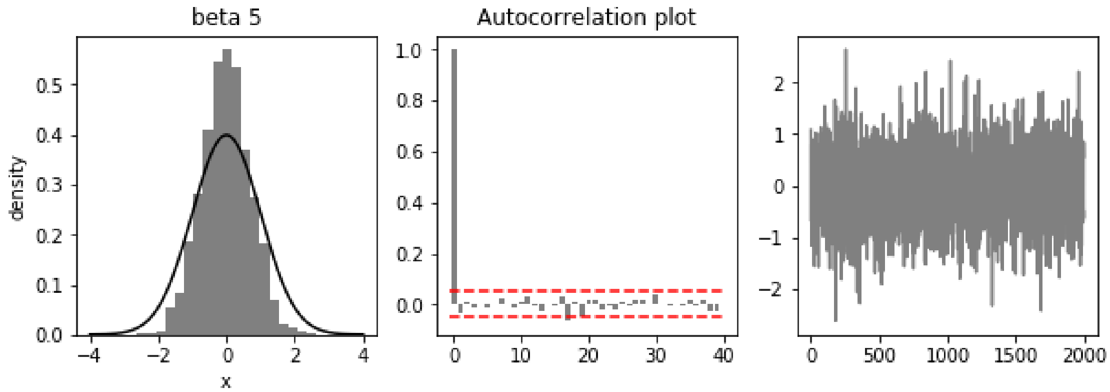

```
In [69]: betaplt(3)
```



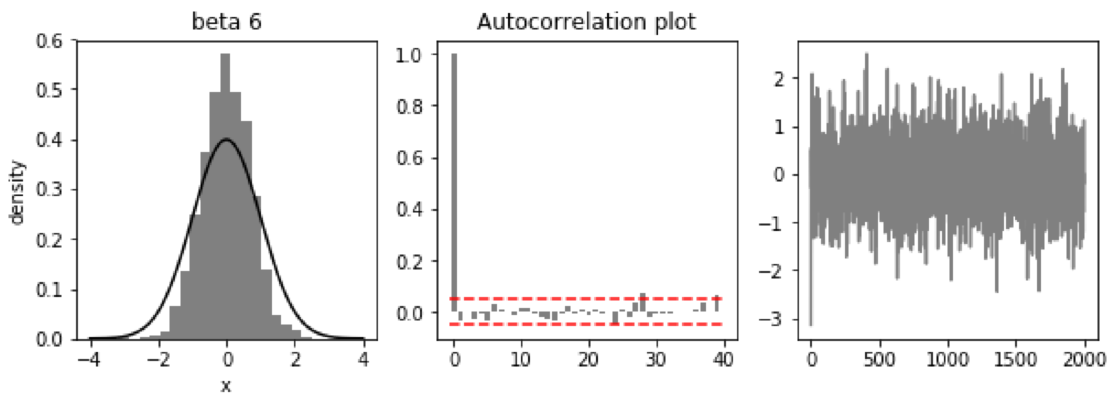

```
In [70]: betaplt(4)
```



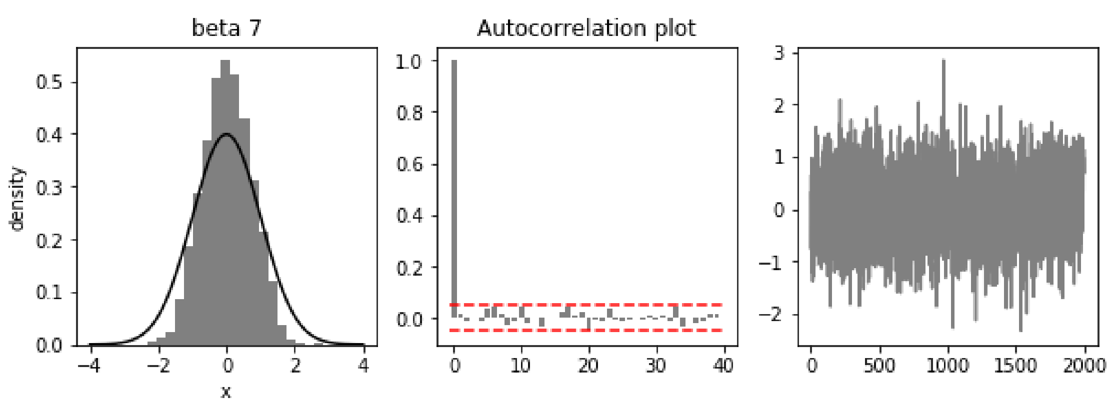
```
In [71]: betaplt(5)
```



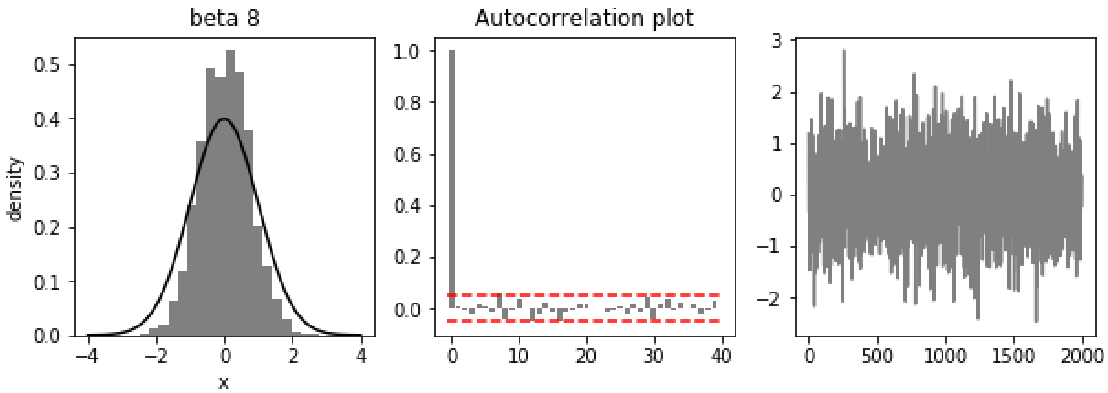

```
In [72]: betaplt(6)
```



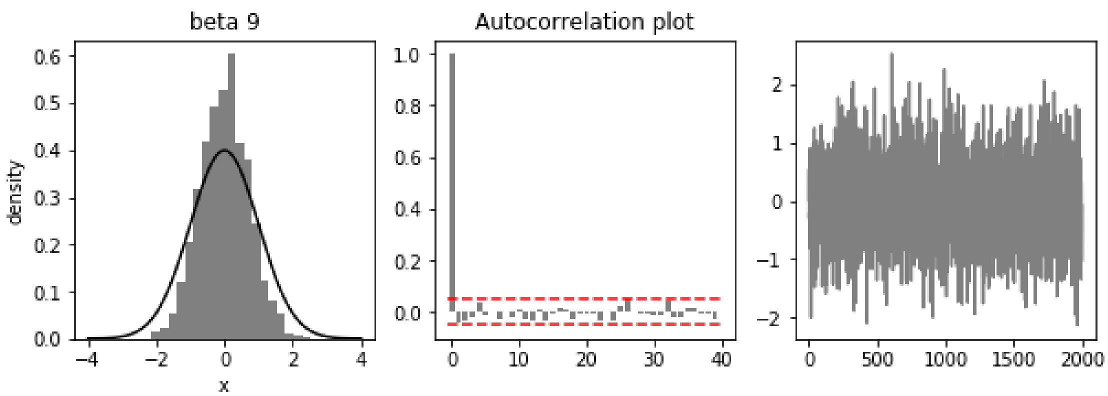

```
In [73]: betaplt(7)
```



```
In [74]: betaplt(8)
```




```
In [75]: betaplt(9)
```




```
In [56]: bdfarray = np.array(bdf)

In [51]: def mfvb(X,y,max_iter=100):

             N,p = X.shape
             a ,b, c, d = [10**(-7)]*4
             a_tilde = np.repeat(a + 0.5, p)
             b_tilde = np.repeat(b,p)
             c_tilde = c + (N+1)/2
             d_tilde = d
```

```
    mu_coeffs = np.repeat(0,p)
    sigma_coeffs = np.diag(np.repeat(1,p))

    for i in range(max_iter):
        expected_coeffs = mu_coeffs
        double_expected_coeffs = sigma_coeffs + product(mu_coeffs)
        diagonal_sigma = np.diag(sigma_coeffs)
        expected_alpha = np.array(list(map(lambda x : a_tilde[x]/b_tilde[x] , np.arange
        log_expected_alpha = np.array(list(map(lambda x : digamma(a_tilde[x])-np.log(b_
        expected_tau = c_tilde / d_tilde
        log_expected_tau = digamma(c_tilde)-np.log(d_tilde)
        sigma_coeffs = np.linalg.inv(np.diag(expected_alpha)+expected_tau*(X.T.dot(X)))
        mu_coeffs =  expected_tau*sigma_coeffs.dot(X.T.dot(y))
        b_tilde = np.array(list(map(lambda x : (diagonal_sigma[x]+mu_coeffs[x]**2)/2 +
        d_tilde = d+0.5*(y.T.dot(y)) - expected_coeffs.T.dot((X.T.dot(y)))+ 0.5*sum(np.
    return mu_coeffs,sigma_coeffs
```

```
In [52]: m,c = mfvb(Z,y)
```

```
In [53]: m
```

```
Out[53]: array([ -0.03524236,  -2.92799365,   5.4149432 ,  -0.01378641,
                 -0.07064352,  -5.46560637,   0.14415652,   0.07018293,
                  9.37113655, -11.89600525,   4.49279968])
```

```
In [91]: plt.figure(figsize=(10, 3))
         plt.subplot(1,3,1)
         hist(s2lst,tlt='s2')
         plt.subplot(1,3,2)
         acf(s2lst)
         plt.subplot(1,3,3)
         plt.plot(s2lst,color='gray')
         plt.show()
```