BayesianInferenceExample

April 5, 2024

1 Example Bayesian Inference with a Two Parameter Model

[22]:
```
import numpy as np
from scipy.stats import ttest_ind
import matplotlib.pyplot as plt
import math
from scipy.stats import truncnorm
from scipy.stats import norm
import random as rnd
from scipy.stats import beta
```

[62]:
```
# Truncated normal: mean mu, sd sigma, truncated between a and b
def randTNorm(a,b,mu,sigma):
    [aStd,bStd]=[(a - mu) / sigma, (b - mu) / sigma]
    return (truncnorm.rvs(aStd, bStd)*sigma+mu)

def pdfTNorm(a,b,mu,sigma,x): # calculate the pdf of x
    return truncnorm.pdf((x-mu)/sigma, (a-mu)/sigma, (b-mu)/sigma)/sigma
def cdfTNorm(a,b,mu,sigma,x):
    return truncnorm.cdf((x-mu)/sigma, (a-mu)/sigma, (b-mu)/sigma)
def round_up_to_even(x):
    return math.ceil(x / 2.) * 2
```

1.1 Simulating the Data and Defining the Liklihood and Priors

Here we consider a 2-state DTDS Markov chain with a transition probability matrix:

$$M = \begin{bmatrix} 1-a & a \\ b & 1-b \end{bmatrix}$$

Suppose we are interested in estimating the values of a and b given a sample trajectory of 50 data points (plus the initial condition which we assume to be in state 0).

[2]:
```
def simData(a,b,npts):
    Mmtrx=np.array([[1-a,a],[b,1-b]])
    xList=np.zeros(npts+1, dtype=int)
    for t in range(1,npts):
        xList[t]=rnd.choices([0,1], weights=Mmtrx[xList[t-1]])[0]
```

```
    return xList
```

Creating one example data set.

[280]:
```
dataTest=simData(0.1,0.8,50)  #A single true data set
nullData=[] #An empty data set used for testing
```

1.1.1 Deifning the likelihood and the prior

Calculating the likleihood and log-likelihood for practice

[15]:
```
def Lik(a,b,data):
    Mmtrx=np.array([[1-a,a],[b,1-b]])
    n=data.size
    out=1
    for t in range(1,n):
        out*=Mmtrx[data[t-1],data[t]]
    return out
```

[16]:
```
Lik(0.1,0.2,dataTest)
```

[16]:
```
3.0237106023979756e-08
```

[21]:
```
math.log(Lik(0.1,0.2,dataTest))
```

[21]:
```
-17.31419599047986
```

[17]:
```
def LnLik(a,b,data):
    Mmtrx=np.array([[1-a,a],[b,1-b]])
    n=data.size
    out=0
    for t in range(1,n):
        out+=np.log(Mmtrx[data[t-1],data[t]])
    return out
```

[20]:
```
LnLik(0.1,0.2,dataTest)
```

[20]:
```
-17.31419599047985
```

Creating a prior for a and b

[23]:
```
def BetaPrior(mean, variance,x):
    # Calculate alpha from mean
    alpha = ((1 - mean) / variance - 1 / mean) * mean**2
    # Calculate beta from alpha and mean
    beta = alpha * (1 / mean - 1)
    return beta.pdf(x, alpha, beta)
```

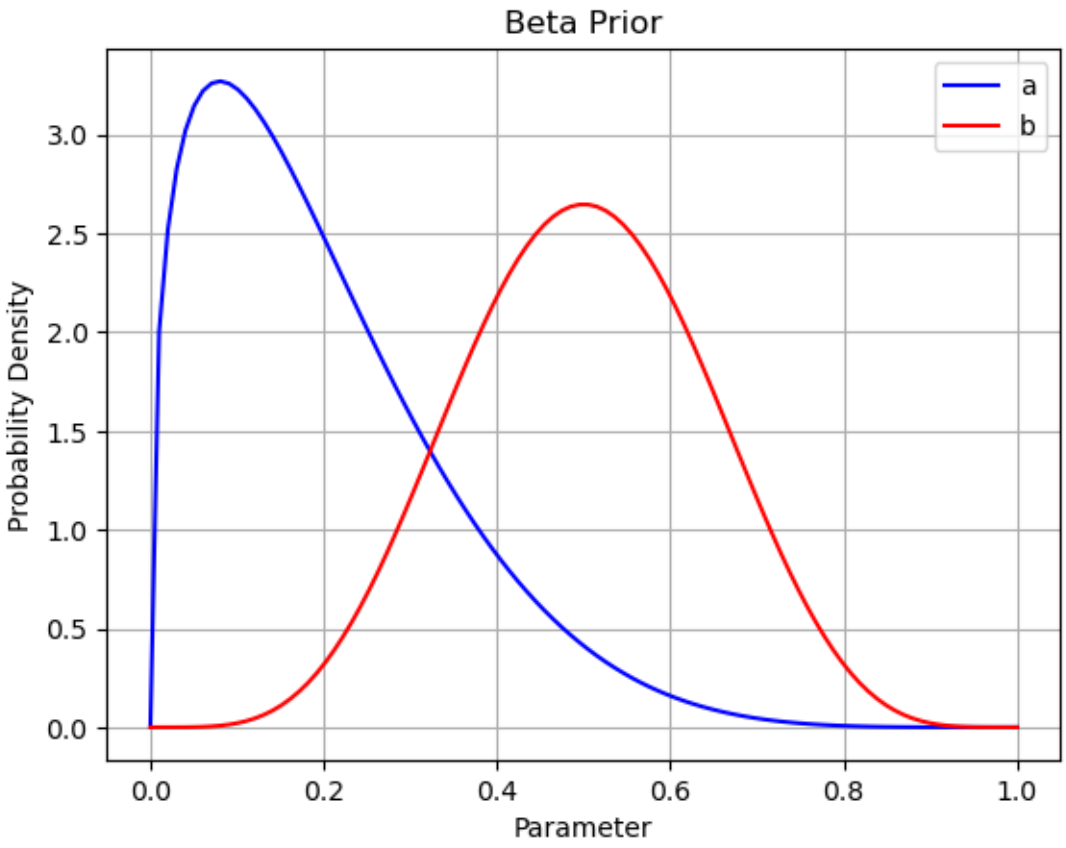
We can sampling randomly from the prior using the function

[170]:
```
def BetaRand(mean, variance):
    # Calculate alpha from mean
    alpha = ((1 - mean) / variance - 1 / mean) * mean**2
    # Calculate beta from alpha and mean
    beta = alpha * (1 / mean - 1)
    return np.random.beta(alpha, beta)
```

Plotting

[284]:
```
# Generate x values
x_values = np.linspace(0, 1, 100)  # 100 points between 0 and 1
# Generate y values
y_values1 = BetaPrior(0.2, 0.02,x_values)
y_values2 = BetaPrior(0.5, 0.02,x_values)

# Plot the function
plt.plot(x_values, y_values1,'b',label='a')
plt.plot(x_values, y_values2,'r',label='b')
plt.xlabel('Parameter')
plt.ylabel('Probability Density')
plt.title('Beta Prior')
plt.legend()
plt.grid(True)
plt.show()
```



1.2 Metropolis-Hastings Algorithm

[271]:
```
class Theta:
    def __init__(self,dataIn, a=0.2,b=0.5,jmpRatio=5):  #Initializing the class
        self.nPar=2
        self.cnt=0
        self.next=None # pointer for linked list
        self.a= a #default parameters
        self.b= b #default parameters
        #Jump Distribution and Prior
        self.aDict={"min":0,"max":1,"jmpVar":1.0/jmpRatio, "PriorMean":0.
↪2,"PriorVar":0.01}
        self.bDict={"min":0,"max":1,"jmpVar":1.0/jmpRatio, "PriorMean":0.
↪5,"PriorVar":0.01}
        #data
        self.data=np.array(dataIn)
        #Calc Probabilities
        self.jump(self)
        self.calcLnLik()
```

```
        self.calcLnPrior()
    def randTheta(self):#Create an initial random theta
        # rnd.seed(10) #set random seed
        # self.a=rnd.uniform(self.aDict["min"],self.aDict["max"])
        # self.b=rnd.uniform(self.bDict["min"],self.bDict["max"])
        self.a=BetaRand(self.aDict["PriorMean"], self.aDict["PriorVar"])
        self.b=BetaRand(self.bDict["PriorMean"], self.aDict["PriorVar"])
        self.jump(self)
        self.calcLnLik()
        self.calcLnPrior()
    def jump(self, parent): #Create a proposed model by jumping from a given␣
↪parent
        self.cnt=parent.cnt+1
        self.a=randTNorm(self.aDict["min"],self.aDict["max"],parent.a,self.
↪aDict["jmpVar"])
        self.b=randTNorm(self.bDict["min"],self.bDict["max"],parent.b,self.
↪bDict["jmpVar"])
        # self.a=rnd.random()
        # self.b=rnd.random()
        self.calcLnLik()
        self.calcLnPrior()
        self.calcJumpProb(parent)
    # Metropolis Hasting probability
    def calcJumpProb(self,parent):
        #Calculate the forward jump probability
        self.jmpProbF=1
        self.jmpProbF*=pdfTNorm(parent.aDict["min"],parent.aDict["max"],parent.
↪a,parent.aDict["jmpVar"],self.a)
        self.jmpProbF*=pdfTNorm(parent.bDict["min"],parent.bDict["max"],parent.
↪b,parent.bDict["jmpVar"],self.b)
        #Calculate the backward jump probability
        self.jmpProbB=1
        self.jmpProbB*=pdfTNorm(self.aDict["min"],self.aDict["max"],self.a,self.
↪aDict["jmpVar"],parent.a)
        self.jmpProbB*=pdfTNorm(self.bDict["min"],self.bDict["max"],self.b,self.
↪bDict["jmpVar"],parent.b)
        #Calculate Acceptance Ratio
        self.lnr=self.lnLik-parent.lnLik+self.lnPrior-parent.lnPrior+math.
↪log(self.jmpProbF)-math.log(self.jmpProbB)
    def calcLnPrior(self):
        self.lnPrior=BetaPrior(self.aDict["PriorMean"], self.
↪aDict["PriorVar"],self.a)*BetaPrior(self.bDict["PriorMean"], self.
↪bDict["PriorVar"],self.b)
        self.lnPrior=math.log(self.lnPrior)
    def calcLnLik(self):
        Mmtrx=np.array([[1-self.a,self.a],[self.b,1-self.b]])
```

```
        n=self.data.size
        out=0
        for t in range(1,n):
            out+=np.log(Mmtrx[self.data[t-1],self.data[t]])
        self.lnLik=out
    def printTheta(self,name,parent):
        nDig=3 #print out to 3 digits
        print(name)
        print('Par a: {}, Par b: {}, Par LogPrior: {}, Par lnLik: {}'.
↪format(round(parent.a,nDig),round(parent.b,nDig),round(parent.
↪lnPrior,nDig),round(parent.lnLik,nDig)))
        print('Forward: {}, Backward: {}, accept prob: {}'.format(round(math.
↪log(self.jmpProbF),nDig),round(math.log(self.jmpProbB),nDig),round(self.
↪lnr,nDig)))
        print('a: {}, b: {}, LogPrior: {}, lnLik: {}'.format(round(self.
↪a,nDig),round(self.b,nDig),round(self.lnPrior,nDig),round(self.lnLik,nDig)))
```

[272]:
```
modelTest=Theta(nullData)
modelTest.printTheta('Test Model',modelTest)
```

```
Test Model
Par a: 0.522, Par b: 0.398, Par LogPrior: -1.545, Par lnLik: 0
Forward: 1.419, Backward: 1.419, accept prob: 0.0
a: 0.522, b: 0.398, LogPrior: -1.545, lnLik: 0
```

[273]:
```
class chain:
    def __init__(self,dataIn):
        self.data=np.array(dataIn)
        self.sampleInitial()
        self.nPar=self.headval.nPar
    def sampleInitial(self):
        theta1=Theta(self.data)
        theta1.randTheta()
        self.headval=theta1
        # self.headval.printTheta("Initial Model",self)
    def MHChain(self,nRep): #simulate chain using the Metropolis-Hastings␣
↪Algorithm
        self.nRep=nRep
        wrkTheta=self.headval
        t=1 # counter of the reps
        acc=0 # conter of accepted jumps
        fail=0 # conter of rejected jumps
        jmpRatio=20 # jump Ratio
        while t<nRep:
            propTheta=Theta(self.data,jmpRatio=jmpRatio) #propose a theta
            propTheta.jump(wrkTheta)
            temp=rnd.random()
```

```
            # if (t+1)%(min(nRep/2,50))==0:
            #       propTheta.printTheta('Step {}'.format(t),wrkTheta)
            if temp<math.exp(propTheta.lnr): #accept
                # print('accept')
                t+=1
                acc+=1
                if (t+1)%(min(nRep/2,50))==0: #priting out progress
                    propTheta.printTheta('Step {}'.format(t),wrkTheta)
                wrkTheta.next=propTheta #link list
                wrkTheta=wrkTheta.next #step along
            else:
                # print('reject')
                fail+=1

    def printChain(self,biRatio): #convert the linked list into an np array of␣
↪parameter values
        out=np.empty((self.nRep,self.nPar),dtype=np.double)
        wrkTheta=self.headval
        for row in range(self.nRep):
            out[row][0]=wrkTheta.a
            out[row][1]=wrkTheta.b
            wrkTheta=wrkTheta.next
        #Handelling burnin (bi)
        self.biRep=round_up_to_even(self.nRep*biRatio)
        #print('chain length: {}, biRep: {}, n: {}\n'.format(len(out),self.
↪biRep,self.n))
        self.psiList=out[self.biRep:] #posterior chain
        self.biList=out[:self.biRep] #burnin chain
    def plotChain(self,var,ax,biRatio): # plot mixing within var 'var' in a␣
↪chain
        self.printChain(biRatio)
        #fig, ax = plt.subplots()
        ax.plot(range(0,self.biRep),self.biList[:,var],color='skyblue',alpha=0.
↪5);
        ax.plot(range(self.biRep,self.nRep),self.psiList[:,var],color='blue');
    def calcKDE(self,biRatio,nBin=1000,a=50): # calculate the kernal density␣
↪estimate from the chain
        self.printChain(biRatio)
        self.KDEList=np.empty((self.nPar,2,nBin),dtype=np.double)
        for var in range(0,self.nPar):
            self.KDEList[var][0]= np.linspace(min(self.psiList[:,var]),␣
↪max(self.psiList[:,var]), nBin)
            dx=self.KDEList[var][0][1]-self.KDEList[var][0][0]
            self.KDEList[var][1] = sum(norm(xi,dx*a).pdf(self.KDEList[var][0])␣
↪for xi in self.psiList[:,var])#smooth out over 50 of the 1000 intervals
            tot=sum(np.multiply(self.KDEList[var][1],dx))
```

```
            self.KDEList[var][1]/=tot
    def findCI(self,var): # find the credible interval
        [x_d,kde]=self.KDEList[var]
        dx=self.KDEList[var][0][1]-self.KDEList[var][0][0]
        #index = np.lexsort((x_d,kde))
        index = np.lexsort((x_d,kde))[::-1] #sort (from largest to smallest) by␣
↪kde then by x_d
        self.maxPost=[x_d[index[0]],kde[index[0]]] #maximum posterior estimate
        temp=np.add.accumulate(kde[index]*dx)
        temp2=index[[i for i,v in enumerate(temp) if v < 0.95]] #find pts in␣
↪credible interval
        indexRev = np.lexsort((kde[temp2],x_d[temp2]))[::-1] #sort (from␣
↪largest to smallest) by x_d then by kde
        return [list(x_d[temp2][indexRev]),list(kde[temp2][indexRev])]
    def kdeHist(self,var,ax,biRatio): # plot hisogram of parameter estimate␣
↪from an individual chain
        self.calcKDE(biRatio)
        [x_d,kde]=self.KDEList[var]
        #plt.fill_between(x_d, density, alpha=0.5) #plot filled smoothed kernel␣
↪density
        ax.hist(self.psiList[:,var], bins=30,density=True, color =␣
↪"skyblue",alpha=0.3) #plot histogram
        ax.plot(x_d,kde,'k') #plot smoothed kernel density line
        temp=self.findCI(var)
        ax.scatter(temp[0],temp[1],color='pink',alpha=1,marker=".") #plot␣
↪credible interval
        ax.plot([self.maxPost[0],self.maxPost[0]],[0, max(kde)*1.2],'r')# Show␣
↪maximum posterior estimate
#       ax.plot([pars.iloc[0,var],pars.iloc[0,var]],[0, max(kde)*1.2],'g')#␣
↪Show true value
        ax.plot(self.psiList[:,var], np.full_like(self.psiList[:
↪,var],-max(kde)*0.05), '|k', markeredgewidth=1) #plot hashes at bottom
        ax.axis([min(x_d), max(x_d), -2*max(kde)*0.05, max(kde)*1.2]);
```

Running the Metropolis-Hasting Chain

1.2.1 Sampling from the prior

[274]:
```
chainTest=chain(nullData)
chainTest.MHChain(2000)
```

```
Step 49
Par a: 0.137, Par b: 0.495, Par LogPrior: 2.752, Par lnLik: 0
Forward: 4.067, Backward: 4.069, accept prob: -0.034
a: 0.128, b: 0.514, LogPrior: 2.721, lnLik: 0
Step 99
Par a: 0.388, Par b: 0.591, Par LogPrior: 0.675, Par lnLik: 0
```

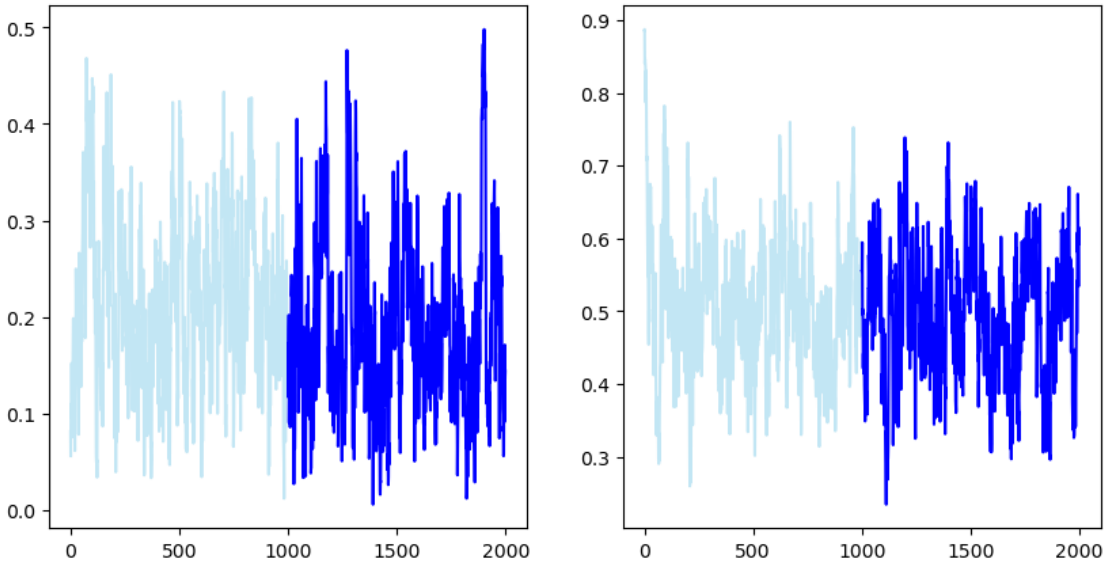
```
Forward: 3.989, Backward: 3.989, accept prob: 0.348
a: 0.36, b: 0.592, LogPrior: 1.023, lnLik: 0
Step 149
Par a: 0.108, Par b: 0.555, Par LogPrior: 2.506, Par lnLik: 0
Forward: 3.004, Backward: 3.031, accept prob: -0.087
a: 0.087, b: 0.482, LogPrior: 2.446, lnLik: 0
Step 199
Par a: 0.157, Par b: 0.622, Par LogPrior: 2.09, Par lnLik: 0
Forward: 2.713, Backward: 2.713, accept prob: -1.392
a: 0.156, b: 0.707, LogPrior: 0.699, lnLik: 0
Step 249
Par a: 0.241, Par b: 0.496, Par LogPrior: 2.466, Par lnLik: 0
Forward: 4.043, Backward: 4.043, accept prob: -0.113
a: 0.255, b: 0.476, LogPrior: 2.353, lnLik: 0
Step 299
Par a: 0.145, Par b: 0.61, Par LogPrior: 2.216, Par lnLik: 0
Forward: 4.075, Backward: 4.074, accept prob: 0.031
a: 0.165, b: 0.607, LogPrior: 2.245, lnLik: 0
Step 349
Par a: 0.14, Par b: 0.402, Par LogPrior: 2.324, Par lnLik: 0
Forward: 4.012, Backward: 4.014, accept prob: -0.28
a: 0.131, b: 0.376, LogPrior: 2.047, lnLik: 0
Step 399
Par a: 0.173, Par b: 0.511, Par LogPrior: 2.745, Par lnLik: 0
Forward: 4.03, Backward: 4.03, accept prob: -0.033
a: 0.188, b: 0.491, LogPrior: 2.712, lnLik: 0
Step 449
Par a: 0.138, Par b: 0.444, Par LogPrior: 2.614, Par lnLik: 0
Forward: 1.768, Backward: 1.765, accept prob: 0.017
a: 0.198, b: 0.535, LogPrior: 2.629, lnLik: 0
Step 499
Par a: 0.284, Par b: 0.545, Par LogPrior: 2.064, Par lnLik: 0
Forward: 4.052, Backward: 4.052, accept prob: 0.178
a: 0.269, b: 0.529, LogPrior: 2.242, lnLik: 0
Step 549
Par a: 0.238, Par b: 0.557, Par LogPrior: 2.344, Par lnLik: 0
Forward: 3.694, Backward: 3.694, accept prob: -0.392
a: 0.285, b: 0.567, LogPrior: 1.952, lnLik: 0
Step 599
Par a: 0.214, Par b: 0.559, Par LogPrior: 2.461, Par lnLik: 0
Forward: 3.828, Backward: 3.828, accept prob: -0.329
a: 0.225, b: 0.598, LogPrior: 2.132, lnLik: 0
Step 649
Par a: 0.144, Par b: 0.615, Par LogPrior: 2.161, Par lnLik: 0
Forward: 3.133, Backward: 3.131, accept prob: 0.31
a: 0.198, b: 0.569, LogPrior: 2.469, lnLik: 0
Step 699
Par a: 0.304, Par b: 0.575, Par LogPrior: 1.726, Par lnLik: 0
```

```
Forward: 3.764, Backward: 3.764, accept prob: -0.376
a: 0.348, b: 0.565, LogPrior: 1.349, lnLik: 0
Step 749
Par a: 0.32, Par b: 0.459, Par LogPrior: 1.748, Par lnLik: 0
Forward: 0.597, Backward: 0.597, accept prob: 0.954
a: 0.191, b: 0.49, LogPrior: 2.703, lnLik: 0
Step 799
Par a: 0.143, Par b: 0.51, Par LogPrior: 2.756, Par lnLik: 0
Forward: 2.905, Backward: 2.907, accept prob: -0.223
a: 0.132, b: 0.432, LogPrior: 2.535, lnLik: 0
Step 849
Par a: 0.289, Par b: 0.416, Par LogPrior: 1.803, Par lnLik: 0
Forward: 3.805, Backward: 3.805, accept prob: 0.412
a: 0.26, b: 0.447, LogPrior: 2.215, lnLik: 0
Step 899
Par a: 0.318, Par b: 0.541, Par LogPrior: 1.769, Par lnLik: 0
Forward: 3.711, Backward: 3.711, accept prob: 0.436
a: 0.272, b: 0.533, LogPrior: 2.205, lnLik: 0
Step 949
Par a: 0.233, Par b: 0.436, Par LogPrior: 2.338, Par lnLik: 0
Forward: 3.172, Backward: 3.172, accept prob: 0.179
a: 0.233, b: 0.506, LogPrior: 2.518, lnLik: 0
Step 999
Par a: 0.119, Par b: 0.562, Par LogPrior: 2.529, Par lnLik: 0
Forward: 3.76, Backward: 3.753, accept prob: -0.164
a: 0.149, b: 0.595, LogPrior: 2.359, lnLik: 0
Step 1049
Par a: 0.125, Par b: 0.581, Par LogPrior: 2.425, Par lnLik: 0
Forward: 2.015, Backward: 2.031, accept prob: 0.133
a: 0.101, b: 0.48, LogPrior: 2.574, lnLik: 0
Step 1099
Par a: 0.141, Par b: 0.49, Par LogPrior: 2.755, Par lnLik: 0
Forward: 3.647, Backward: 3.645, accept prob: -0.117
a: 0.169, b: 0.448, LogPrior: 2.636, lnLik: 0
Step 1149
Par a: 0.273, Par b: 0.403, Par LogPrior: 1.829, Par lnLik: 0
Forward: 2.532, Backward: 2.532, accept prob: -0.802
a: 0.362, b: 0.412, LogPrior: 1.028, lnLik: 0
Step 1199
Par a: 0.223, Par b: 0.664, Par LogPrior: 1.318, Par lnLik: 0
Forward: 2.966, Backward: 2.966, accept prob: -1.702
a: 0.242, b: 0.739, LogPrior: -0.384, lnLik: 0
Step 1249
Par a: 0.115, Par b: 0.329, Par LogPrior: 1.309, Par lnLik: 0
Forward: 1.243, Backward: 1.402, accept prob: 0.131
a: 0.051, b: 0.431, LogPrior: 1.598, lnLik: 0
Step 1299
Par a: 0.077, Par b: 0.545, Par LogPrior: 2.245, Par lnLik: 0
```

```
Forward: 3.907, Backward: 3.946, accept prob: -0.151
a: 0.065, b: 0.508, LogPrior: 2.132, lnLik: 0
Step 1349
Par a: 0.257, Par b: 0.414, Par LogPrior: 2.036, Par lnLik: 0
Forward: 3.998, Backward: 3.998, accept prob: -0.306
a: 0.28, b: 0.399, LogPrior: 1.73, lnLik: 0
Step 1399
Par a: 0.215, Par b: 0.732, Par LogPrior: -0.05, Par lnLik: 0
Forward: 0.51, Backward: 0.51, accept prob: 2.116
a: 0.236, b: 0.598, LogPrior: 2.066, lnLik: 0
Step 1449
Par a: 0.171, Par b: 0.503, Par LogPrior: 2.753, Par lnLik: 0
Forward: 3.783, Backward: 3.783, accept prob: -0.084
a: 0.165, b: 0.546, LogPrior: 2.669, lnLik: 0
Step 1499
Par a: 0.241, Par b: 0.497, Par LogPrior: 2.472, Par lnLik: 0
Forward: 3.944, Backward: 3.944, accept prob: 0.116
a: 0.217, b: 0.519, LogPrior: 2.588, lnLik: 0
Step 1549
Par a: 0.261, Par b: 0.579, Par LogPrior: 2.056, Par lnLik: 0
Forward: 2.808, Backward: 2.808, accept prob: -0.343
a: 0.209, b: 0.642, LogPrior: 1.713, lnLik: 0
Step 1599
Par a: 0.256, Par b: 0.318, Par LogPrior: 0.81, Par lnLik: 0
Forward: 3.16, Backward: 3.16, accept prob: 1.03
a: 0.204, b: 0.366, LogPrior: 1.84, lnLik: 0
Step 1649
Par a: 0.13, Par b: 0.471, Par LogPrior: 2.7, Par lnLik: 0
Forward: 3.537, Backward: 3.534, accept prob: 0.05
a: 0.153, b: 0.522, LogPrior: 2.746, lnLik: 0
Step 1699
Par a: 0.147, Par b: 0.387, Par LogPrior: 2.187, Par lnLik: 0
Forward: 3.456, Backward: 3.454, accept prob: 0.152
a: 0.2, b: 0.413, LogPrior: 2.337, lnLik: 0
Step 1749
Par a: 0.16, Par b: 0.599, Par LogPrior: 2.325, Par lnLik: 0
Forward: 4.134, Backward: 4.135, accept prob: 0.049
a: 0.152, b: 0.594, LogPrior: 2.375, lnLik: 0
Step 1799
Par a: 0.185, Par b: 0.613, Par LogPrior: 2.146, Par lnLik: 0
Forward: 3.276, Backward: 3.282, accept prob: -0.352
a: 0.125, b: 0.642, LogPrior: 1.8, lnLik: 0
Step 1849
Par a: 0.132, Par b: 0.346, Par LogPrior: 1.639, Par lnLik: 0
Forward: 3.571, Backward: 3.567, accept prob: -0.649
a: 0.171, b: 0.307, LogPrior: 0.986, lnLik: 0
Step 1899
Par a: 0.483, Par b: 0.573, Par LogPrior: -0.598, Par lnLik: 0
```

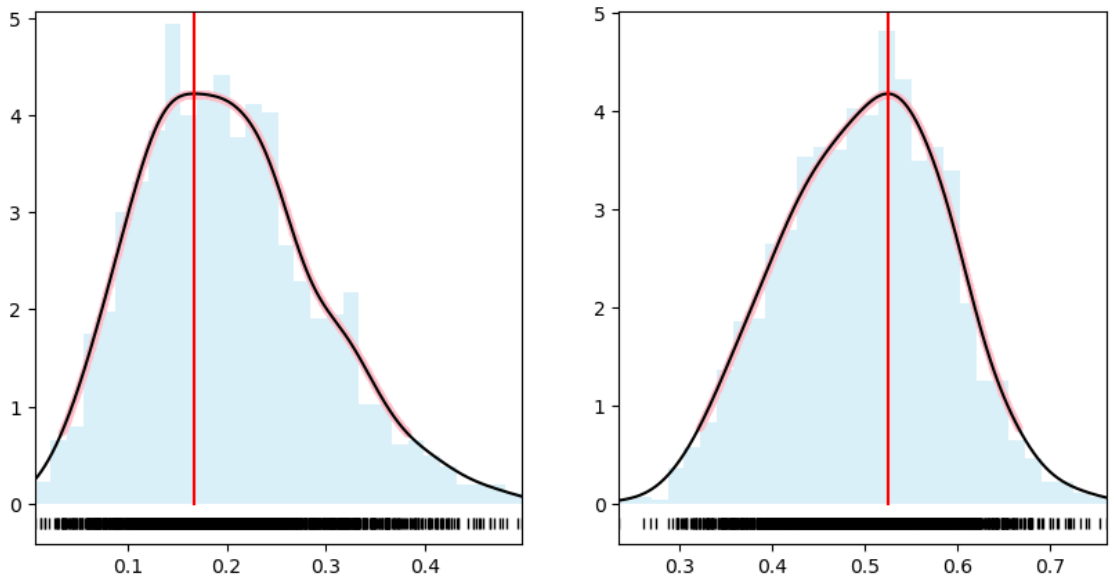
```
Forward: 3.87, Backward: 3.87, accept prob: 0.649
a: 0.449, b: 0.557, LogPrior: 0.051, lnLik: 0
Step 1949
Par a: 0.316, Par b: 0.516, Par LogPrior: 1.855, Par lnLik: 0
Forward: 2.877, Backward: 2.877, accept prob: -0.541
a: 0.331, b: 0.595, LogPrior: 1.314, lnLik: 0
Step 1999
Par a: 0.171, Par b: 0.615, Par LogPrior: 2.152, Par lnLik: 0
Forward: 3.876, Backward: 3.879, accept prob: 0.108
a: 0.136, b: 0.604, LogPrior: 2.264, lnLik: 0
```

[275]:
```
fig, ax0 =  plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
for v in range(2):
    chainTest.plotChain(v,ax0[v],0.5)
```



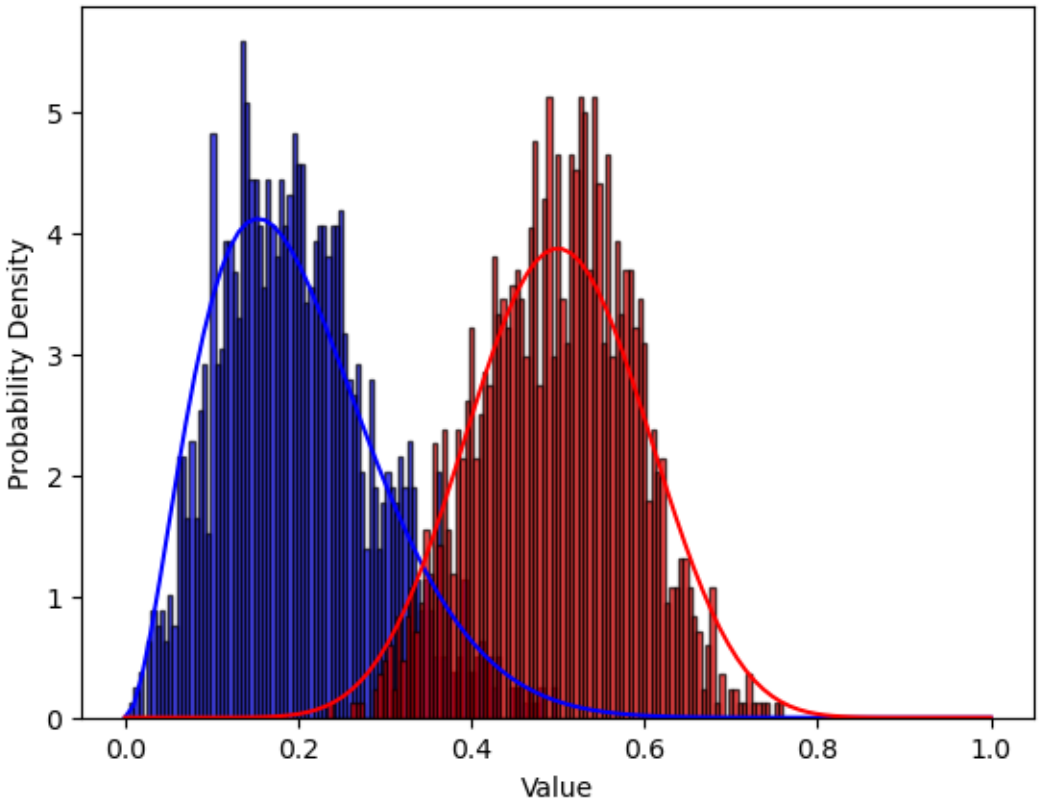
Plotting the posteriors

[276]:
```
fig, ax1 =  plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
chainTest.calcKDE(0.2,nBin=1000,a=50)
for v in range(2):
    chainTest.findCI(v)
    chainTest.kdeHist(v,ax1[v],0.2)
```



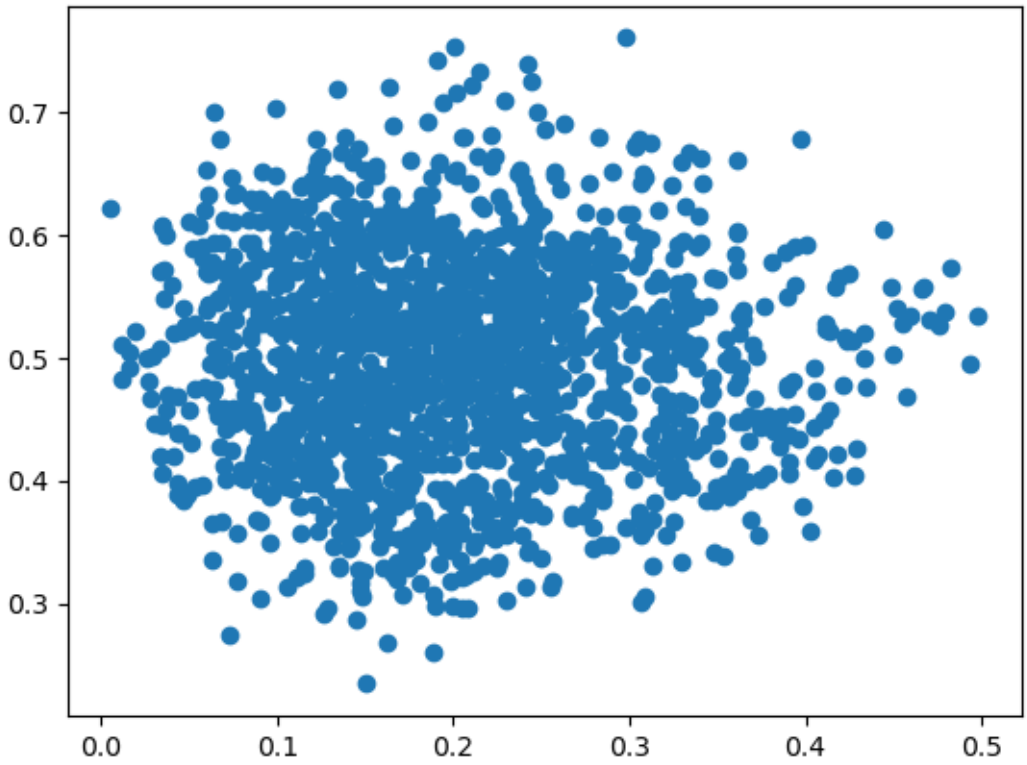
[277]:
```
# Generate x values
x_values = np.linspace(0, 1, 100)  # 100 points between 0 and 1
# Generate y values
y_values1 = BetaPrior(0.2, 0.01,x_values)
y_values2 = BetaPrior(0.5, 0.01,x_values)

# Plot the function
plt.plot(x_values, y_values1,'b')
plt.plot(x_values, y_values2,'r')
plt.hist(chainTest.psiList[:,0], bins=100, density=True, alpha=0.7,␣
 ↪color='blue', edgecolor='black')
plt.hist(chainTest.psiList[:,1], bins=100, density=True, alpha=0.7,␣
 ↪color='red', edgecolor='black')
plt.xlabel('Value')
plt.ylabel('Probability Density');
```



[278]:
```
plt.scatter(chainTest.psiList[:,0],chainTest.psiList[:,1])
```

[278]:
```
<matplotlib.collections.PathCollection at 0x7f1d0423ec50>
```



1.3 Sampling from posterior

[279]:
```
chainTest=chain(dataTest)
chainTest.MHChain(2000)
```

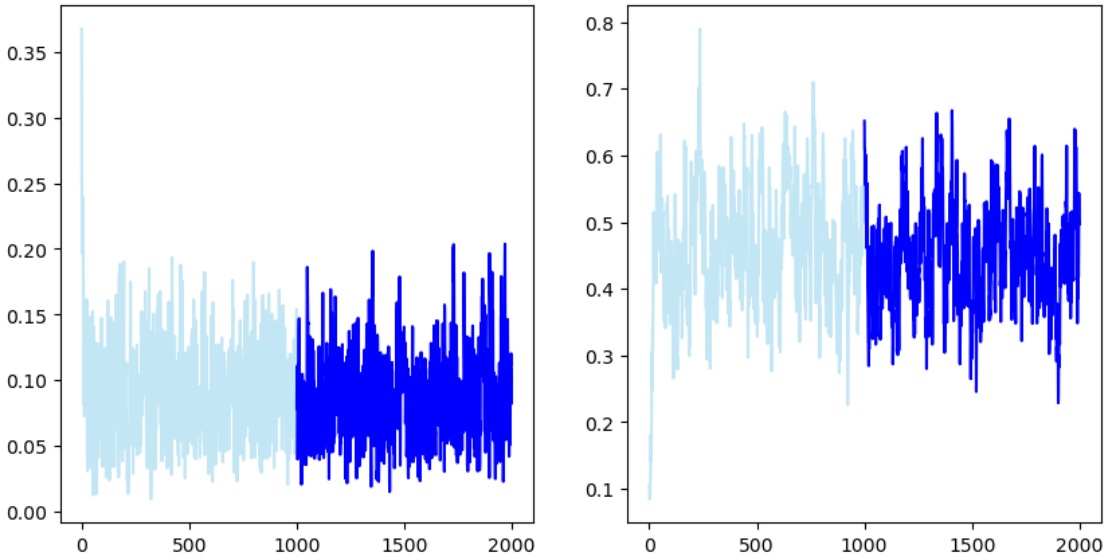
```
Step 49
Par a: 0.05, Par b: 0.54, Par LogPrior: 1.717, Par lnLik: -12.482
Forward: 4.235, Backward: 4.25, accept prob: 0.075
a: 0.047, b: 0.518, LogPrior: 1.699, lnLik: -12.374
Step 99
Par a: 0.049, Par b: 0.37, Par LogPrior: 0.996, Par lnLik: -11.979
Forward: 3.84, Backward: 3.945, accept prob: -0.33
a: 0.034, b: 0.417, LogPrior: 0.912, lnLik: -12.12
Step 149
Par a: 0.118, Par b: 0.461, Par LogPrior: 2.625, Par lnLik: -13.574
Forward: 3.681, Backward: 3.714, accept prob: 0.486
a: 0.087, b: 0.423, LogPrior: 2.198, lnLik: -12.628
Step 199
Par a: 0.19, Par b: 0.511, Par LogPrior: 2.703, Par lnLik: -16.394
Forward: 1.617, Backward: 1.662, accept prob: 2.901
a: 0.086, b: 0.552, LogPrior: 2.329, lnLik: -13.076
Step 249
```

```
Par a: 0.065, Par b: 0.512, Par LogPrior: 2.139, Par lnLik: -12.502
Forward: 3.235, Backward: 3.204, accept prob: 0.135
a: 0.075, b: 0.442, LogPrior: 2.156, lnLik: -12.416
Step 299
Par a: 0.134, Par b: 0.399, Par LogPrior: 2.291, Par lnLik: -13.928
Forward: 3.957, Backward: 3.956, accept prob: -0.626
a: 0.144, b: 0.37, LogPrior: 1.988, lnLik: -14.253
Step 349
Par a: 0.102, Par b: 0.543, Par LogPrior: 2.517, Par lnLik: -13.439
Forward: 4.053, Backward: 4.05, accept prob: 0.126
a: 0.105, b: 0.519, LogPrior: 2.605, lnLik: -13.403
Step 399
Par a: 0.103, Par b: 0.467, Par LogPrior: 2.56, Par lnLik: -13.154
Forward: 2.972, Backward: 3.245, accept prob: -0.88
a: 0.033, b: 0.434, LogPrior: 0.972, lnLik: -12.174
Step 449
Par a: 0.065, Par b: 0.431, Par LogPrior: 1.938, Par lnLik: -12.234
Forward: 4.231, Backward: 4.277, accept prob: -0.162
a: 0.055, b: 0.429, LogPrior: 1.697, lnLik: -12.109
Step 499
Par a: 0.07, Par b: 0.348, Par LogPrior: 1.159, Par lnLik: -12.182
Forward: 3.283, Backward: 3.204, accept prob: -0.365
a: 0.125, b: 0.389, LogPrior: 2.167, lnLik: -13.634
Step 549
Par a: 0.081, Par b: 0.388, Par LogPrior: 1.819, Par lnLik: -12.426
Forward: 4.105, Backward: 4.079, accept prob: -0.015
a: 0.095, b: 0.405, LogPrior: 2.141, lnLik: -12.788
Step 599
Par a: 0.062, Par b: 0.428, Par LogPrior: 1.853, Par lnLik: -12.181
Forward: 3.796, Backward: 3.755, accept prob: 0.135
a: 0.074, b: 0.475, LogPrior: 2.263, lnLik: -12.497
Step 649
Par a: 0.094, Par b: 0.551, Par LogPrior: 2.422, Par lnLik: -13.273
Forward: 3.589, Backward: 3.571, accept prob: -1.0
a: 0.112, b: 0.602, LogPrior: 2.19, lnLik: -14.059
Step 699
Par a: 0.121, Par b: 0.434, Par LogPrior: 2.509, Par lnLik: -13.576
Forward: 3.721, Backward: 3.724, accept prob: -0.118
a: 0.114, b: 0.387, LogPrior: 2.099, lnLik: -13.28
Step 749
Par a: 0.104, Par b: 0.454, Par LogPrior: 2.522, Par lnLik: -13.145
Forward: 4.059, Backward: 4.079, accept prob: 0.201
a: 0.089, b: 0.436, LogPrior: 2.302, lnLik: -12.705
Step 799
Par a: 0.07, Par b: 0.469, Par LogPrior: 2.19, Par lnLik: -12.413
Forward: 4.113, Backward: 4.211, accept prob: -0.357
a: 0.049, b: 0.456, LogPrior: 1.664, lnLik: -12.146
Step 849
```

```
Par a: 0.086, Par b: 0.441, Par LogPrior: 2.292, Par lnLik: -12.637
Forward: 3.499, Backward: 3.458, accept prob: -1.229
a: 0.135, b: 0.408, LogPrior: 2.374, lnLik: -13.988
Step 899
Par a: 0.109, Par b: 0.376, Par LogPrior: 1.948, Par lnLik: -13.132
Forward: 1.689, Backward: 1.874, accept prob: 0.439
a: 0.046, b: 0.467, LogPrior: 1.615, lnLik: -12.176
Step 949
Par a: 0.097, Par b: 0.605, Par LogPrior: 2.071, Par lnLik: -13.683
Forward: 3.629, Backward: 3.799, accept prob: -0.069
a: 0.046, b: 0.594, LogPrior: 1.275, lnLik: -12.785
Step 999
Par a: 0.129, Par b: 0.544, Par LogPrior: 2.647, Par lnLik: -14.239
Forward: 3.151, Backward: 3.147, accept prob: -1.548
a: 0.149, b: 0.612, LogPrior: 2.202, lnLik: -15.345
Step 1049
Par a: 0.086, Par b: 0.407, Par LogPrior: 2.064, Par lnLik: -12.57
Forward: 2.906, Backward: 2.863, accept prob: -1.926
a: 0.163, b: 0.431, LogPrior: 2.551, lnLik: -15.025
Step 1099
Par a: 0.087, Par b: 0.473, Par LogPrior: 2.433, Par lnLik: -12.77
Forward: 4.147, Backward: 4.161, accept prob: 0.032
a: 0.08, b: 0.487, LogPrior: 2.373, lnLik: -12.663
Step 1149
Par a: 0.101, Par b: 0.375, Par LogPrior: 1.89, Par lnLik: -12.91
Forward: 2.032, Backward: 2.054, accept prob: 0.672
a: 0.086, b: 0.478, LogPrior: 2.427, lnLik: -12.752
Step 1199
Par a: 0.125, Par b: 0.438, Par LogPrior: 2.549, Par lnLik: -13.715
Forward: 3.281, Backward: 3.399, accept prob: 0.688
a: 0.06, b: 0.426, LogPrior: 1.787, lnLik: -12.148
Step 1249
Par a: 0.038, Par b: 0.399, Par LogPrior: 0.921, Par lnLik: -12.041
Forward: 3.927, Backward: 3.737, accept prob: 0.991
a: 0.077, b: 0.428, LogPrior: 2.113, lnLik: -12.434
Step 1299
Par a: 0.075, Par b: 0.456, Par LogPrior: 2.219, Par lnLik: -12.453
Forward: 3.444, Backward: 3.471, accept prob: -0.18
a: 0.066, b: 0.518, LogPrior: 2.156, lnLik: -12.545
Step 1349
Par a: 0.135, Par b: 0.442, Par LogPrior: 2.598, Par lnLik: -14.056
Forward: 3.278, Backward: 3.333, accept prob: 1.138
a: 0.079, b: 0.478, LogPrior: 2.346, lnLik: -12.611
Step 1399
Par a: 0.047, Par b: 0.501, Par LogPrior: 1.714, Par lnLik: -12.301
Forward: 3.986, Backward: 3.98, accept prob: -0.244
a: 0.048, b: 0.543, LogPrior: 1.657, lnLik: -12.495
Step 1449
```

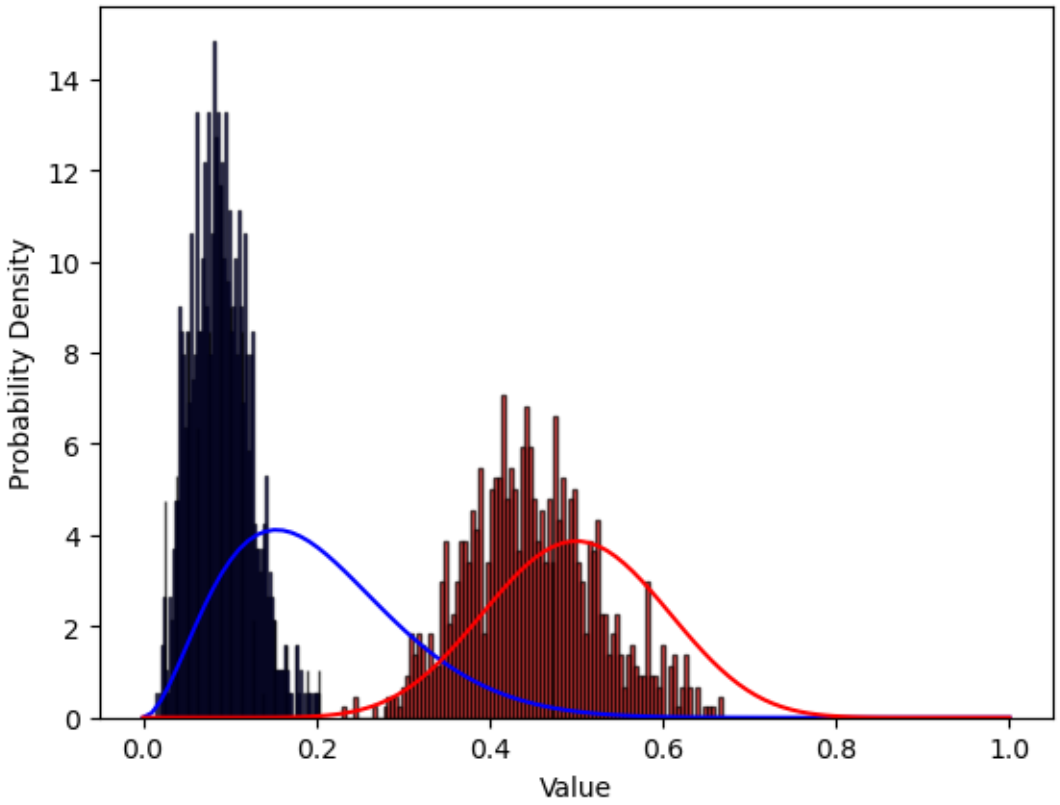
```
Par a: 0.084, Par b: 0.39, Par LogPrior: 1.883, Par lnLik: -12.506
Forward: 4.184, Backward: 4.169, accept prob: -0.121
a: 0.093, b: 0.387, LogPrior: 1.95, lnLik: -12.71
Step 1499
Par a: 0.12, Par b: 0.332, Par LogPrior: 1.39, Par lnLik: -13.421
Forward: 3.851, Backward: 3.858, accept prob: -0.468
a: 0.108, b: 0.295, LogPrior: 0.611, lnLik: -13.103
Step 1549
Par a: 0.075, Par b: 0.399, Par LogPrior: 1.853, Par lnLik: -12.333
Forward: 4.188, Backward: 4.217, accept prob: 0.043
a: 0.066, b: 0.408, LogPrior: 1.79, lnLik: -12.2
Step 1599
Par a: 0.112, Par b: 0.588, Par LogPrior: 2.312, Par lnLik: -13.965
Forward: 3.145, Backward: 3.368, accept prob: -0.133
a: 0.04, b: 0.583, LogPrior: 1.165, lnLik: -12.728
Step 1649
Par a: 0.073, Par b: 0.481, Par LogPrior: 2.266, Par lnLik: -12.507
Forward: 4.032, Backward: 3.976, accept prob: -0.328
a: 0.104, b: 0.477, LogPrior: 2.591, lnLik: -13.216
Step 1699
Par a: 0.075, Par b: 0.437, Par LogPrior: 2.14, Par lnLik: -12.415
Forward: 3.488, Backward: 3.493, accept prob: -0.405
a: 0.073, b: 0.376, LogPrior: 1.593, lnLik: -12.268
Step 1749
Par a: 0.037, Par b: 0.447, Par LogPrior: 1.225, Par lnLik: -12.155
Forward: 4.204, Backward: 4.063, accept prob: 0.806
a: 0.061, b: 0.469, LogPrior: 2.018, lnLik: -12.282
Step 1799
Par a: 0.088, Par b: 0.492, Par LogPrior: 2.47, Par lnLik: -12.854
Forward: 3.957, Backward: 3.935, accept prob: -0.235
a: 0.105, b: 0.462, LogPrior: 2.56, lnLik: -13.201
Step 1849
Par a: 0.133, Par b: 0.448, Par LogPrior: 2.623, Par lnLik: -14.004
Forward: 2.121, Backward: 2.227, accept prob: 0.851
a: 0.063, b: 0.52, LogPrior: 2.081, lnLik: -12.505
Step 1899
Par a: 0.167, Par b: 0.31, Par LogPrior: 1.05, Par lnLik: -15.083
Forward: 4.041, Backward: 4.042, accept prob: 0.738
a: 0.158, b: 0.332, LogPrior: 1.456, lnLik: -14.75
Step 1949
Par a: 0.053, Par b: 0.43, Par LogPrior: 1.647, Par lnLik: -12.096
Forward: 4.259, Backward: 4.192, accept prob: 0.242
a: 0.068, b: 0.434, LogPrior: 2.01, lnLik: -12.284
Step 1999
Par a: 0.082, Par b: 0.544, Par LogPrior: 2.321, Par lnLik: -12.951
Forward: 3.623, Backward: 3.586, accept prob: -0.1
a: 0.108, b: 0.496, LogPrior: 2.642, lnLik: -13.409
```

[281]:
```
fig, ax0 =  plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
for v in range(2):
    chainTest.plotChain(v,ax0[v],0.5)
```



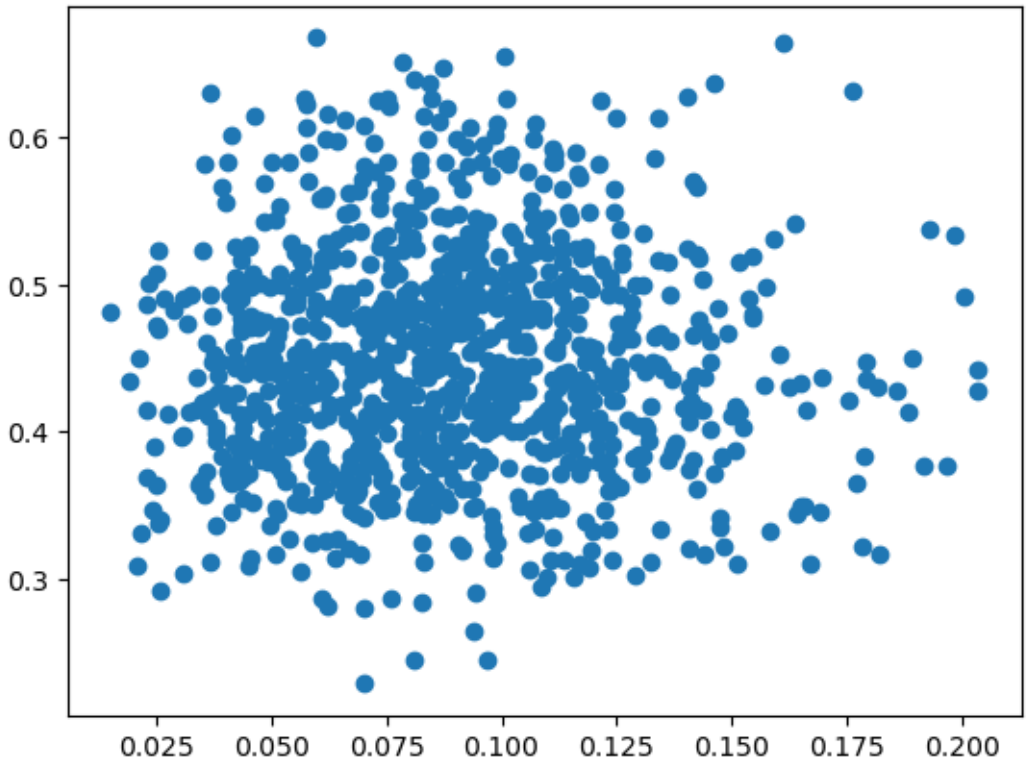
[282]:
```
# Generate x values
x_values = np.linspace(0, 1, 100)  # 100 points between 0 and 1
# Generate y values
y_values1 = BetaPrior(0.2, 0.01,x_values)
y_values2 = BetaPrior(0.5, 0.01,x_values)

# Plot the function
plt.plot(x_values, y_values1,'b')
plt.plot(x_values, y_values2,'r')
plt.hist(chainTest.psiList[:,0], bins=100, density=True, alpha=0.7,␣
 ↪color='blue', edgecolor='black')
plt.hist(chainTest.psiList[:,1], bins=100, density=True, alpha=0.7,␣
 ↪color='red', edgecolor='black')
plt.xlabel('Value')
plt.ylabel('Probability Density');
```



[287]:
```
plt.scatter(chainTest.psiList[:,0],chainTest.psiList[:,1])
correlation = np.corrcoef(chainTest.psiList[:,0], chainTest.psiList[:,1])[0, 1]
print(correlation)
```

```
0.003321782888233707
```


0.6
0.5
0.4
0.3
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200