Source Codes for Various Simulation Studies in Chapter 4

I.'''Python Code for Case 1,2,3'''

```
from numpy import *

from scipy import *

from math import *

from random import *

from statistics import *

import matplotlib.pyplot as plt


''' 3 point process with a specific density function on [0,1.5] * [0,1]'''


#initial position

x_1 = 1.5*random()

y_1 = random()

x_2 = 1.5*random()

y_2 = random()

x_3 = 1.5*random()

y_3 = random()
```

```
ini_pos = array([x_1,y_1,x_2,y_2,x_3,y_3],float)


#Now define pdf

def h(x, A,B,C):

   #x is the array for positions of the 3 particles

   sum_dis = float(sqrt((x[0]-x[2])**2+(x[1]-x[3])**2) + sqrt((x[0]-x[4])**2+(x[1]-x[5])**2) + sqrt((x[2]-
x[4])**2+(x[3]-x[5])**2))

   sum_x = float(x[0]+x[2]+x[4])

   value = A*sum_dis + 0.04*B*(1/float(sqrt((x[0]-x[2])**2+(x[1]-x[3])**2))+1/float(sqrt((x[0]-
x[4])**2+(x[1]-x[5])**2))+1/float(sqrt((x[2]-x[4])**2+(x[3]-x[5])**2))) + C*sum_x

   return value


#def pi(x,A,B,C):

   ##target distribution up to a constant

   #if 0<=x[0]<=1.5 and 0<=x[2]<=1.5 and 0<=x[4]<=1.5 and 0<=x[1]<=1 and 0<=x[3]<=1 and 0<=x[5]<=1:

      ##restricting to pink region

      #return exp(-h(x,A,B,C))

   #else:

      #return 0


#We commented out the definition of pdf because we are doing the rejection/acceptance on log scale

#to avoid numerical errors

'''We use symmetric uniform proposal in [-0.1,0.1] * [-0.1,0.1] without adaptation using metropolis
within gibbs method with optimal scaling'''


def run(step,A,B,C):
```

```
x = ini_pos
reject_times = 0
accept_times = 0

x_1_pos = []
y_1_pos = []
time = []

i = 0
while i <= step:
    time.append(i)
    x_1_pos.append(x[0])
    y_1_pos.append(x[1])



    dummy = [0,0,0,0,0,0]
    #pick a point at random
    a = randint(1,3)
    r1 = uniform(-0.1,0.1)
    r2 = uniform(-0.1,0.1)
    x_pos = a*2-2
    dummy[x_pos] = r1
    dummy[x_pos+1] = r2

    increment = array(dummy)
```

```
        proposal = x + increment


        #If the proposal is in the region
        if (0<=proposal[0]<=1.5 and 0<=proposal[2]<=1.5 and 0<=proposal[4]<=1.5 and 0<=proposal[1]<=1
and 0<=proposal[3]<=1 and 0<=proposal[5]<=1):
            accept = h(x,A,B,C) - h(proposal,A,B,C)


            #determine the acceptance probability on log scale
            k = random()
            if log(k) < accept:
                x = proposal
                accept_times += 1
            else:
                x = x #proposal is rejected
                reject_times += 1
        else:
            x = x #proposal is rejected if the proposal is not in the region
            reject_times += 1
        i +=1
    return x_1_pos,y_1_pos,time,1-(reject_times/float(step)), accept_times, reject_times



#To generate the trace plot


#plt.scatter(a[0], a[1], s=2, c='b', marker='o', cmap=None, norm=None,vmin=None, vmax=None,
alpha=1.0, linewidths=None,verts=None)
```

```
#plt.plot(a[2],a[0], "b")

#plt.plot(a[2],a[1], "g")

#plt.show()


#To estimate Var(1/n \sum g(Xi)) where g is taken as the x coordinate of the first particle

def get_variance(step,A,B,C):

    k = 1

    mean_0 = []

    while k <= 100:

        a = run(step,A,B,C)

        mean_0.append(mean(a[0]))

        k += 1

    return (std(mean_0))**2


def get_corr_x0_xi(l,i):

    x=[]

    y=[]

    k = 0

    while k + i < len(l):

        x.append(l[k])

        y.append(l[k+i])

        k += 1

    return correlation(x,y)


#To get the graph of auto correlation time
```

```
def get_corr_graph(l,step):
    a = []
    b = []
    i = 1
    while i <= step:
        a.append(get_corr_x0_xi(l,i))
        b.append(i)
        i += 1
    return a,b




def get_iact(step,A,B,C):
    '''Here g is taken as the first coordinate of the vector'''

    x1_final = run(step,A,B,C)[0]
    sum = 0
    i = 1
    k = 0
    while i <= step and k <= 1:
        q = get_corr_x0_xi(x1_final,i)
        if q < 0:
            k += 1
        sum += q
        i += 1
```

```
    return 1 + 2*sum, i

#set the cutoff value manually
def get_iact_manual(step,A,B,C,cutoff):
    '''Here g is taken as the first coordinate of the vector'''

    x1_final = run(step,A,B,C)[0]
    sum = 0
    i = 1
    while i <= cutoff:
        q = get_corr_x0_xi(x1_final,i)
        sum += q
        i += 1
    return 1 + 2*sum, i

#use output list directly without computing
def get_withop(step,l):
    '''Here g is taken as the first coordinate of the vector'''

    x1_final = l
    sum = 0
    i = 1
    k = 0
    while i <= step and k <= 1:
        q = get_corr_x0_xi(x1_final,i)
```

```
        if q < 0:
            k += 1
        sum += q
        i += 1
    return 1 + 2*sum, i


#Run the chain for 300,000 steps
#a = run(300000,200,200,0)
#taug = get_withop(50000,a[0])
#var1 = std(a[0])**2
#var2 = get_variance(300000,200,200,0)
#print "tau_g", taug[0]
#print "lag",taug[1]
#print "var(g(X))", var1
#print "other variance", var2
#print "ratio", 300000*var2/float(taug[0])



#c = get_corr_graph(a[0],50000)
#plt.plot(c[1],c[0], "b")
#plt.show()
```

II.'''Python Code for Case4'''

```
from numpy import *
```

```
from scipy import *

from math import *

from random import *

from statistics import *

import matplotlib.pyplot as plt


''' 3 point process with a specific density function on [0,1.5] * [0,1]'''


#initial position

x_1 = 1.5*random()

y_1 = random()

x_2 = 1.5*random()

y_2 = random()

x_3 = 1.5*random()

y_3 = random()


ini_pos = array([x_1,y_1,x_2,y_2,x_3,y_3],float)


#Now define pdf

def h(x, A,B,C):

    #x is the array for positions of the 3 particles

    sum_dis = float(sqrt((x[0]-x[2])**2+(x[1]-x[3])**2) + sqrt((x[0]-x[4])**2+(x[1]-x[5])**2) + sqrt((x[2]-
x[4])**2+(x[3]-x[5])**2))

    sum_x = float(x[0]+x[2]+x[4])

    value = A*sum_dis + 0.04*B*(1/float(sqrt((x[0]-x[2])**2+(x[1]-x[3])**2))+1/float(sqrt((x[0]-
x[4])**2+(x[1]-x[5])**2))+1/float(sqrt((x[2]-x[4])**2+(x[3]-x[5])**2))) + C*sum_x
```

```
    return value


#def pi(x,A,B,C):
    ##target distribution up to a constant
    #if 0<=x[0]<=1.5 and 0<=x[2]<=1.5 and 0<=x[4]<=1.5 and 0<=x[1]<=1 and 0<=x[3]<=1 and 0<=x[5]<=1:
        ##restricting to pink region
        #return exp(-h(x,A,B,C))
    #else:
        #return 0


#We commented out the definition of pdf because we are doing the rejection/acceptance on log scale
#to avoid numerical errors
'''Metropolis within Gibbs Adaptive Algorithm to Target Acceptance Rate'''


def run(step,A,B,C,rate_set):
    #rate set is the target acceptance rate
    x = ini_pos
    reject_times = 0
    accept_times = 0

    x_1_pos = []
    y_1_pos = []
    time = []
    length = 0.1
```

```
    i = 0

    while i <= step:

        time.append(i)

        x_1_pos.append(x[0])

        y_1_pos.append(x[1])




        dummy = [0,0,0,0,0,0]

        #pick a point at random

        a = randint(1,3)

        r1 = uniform(-length,length)

        r2 = uniform(-length,length)

        x_pos = a*2-2

        dummy[x_pos] = r1

        dummy[x_pos+1] = r2


        increment = array(dummy)

        proposal = x + increment


        if (0<=proposal[0]<=1.5 and 0<=proposal[2]<=1.5 and 0<=proposal[4]<=1.5 and 0<=proposal[1]<=1 
and 0<=proposal[3]<=1 and 0<=proposal[5]<=1):

            accept = h(x,A,B,C) - h(proposal,A,B,C)


            k = random()

            if log(k) < accept:

                x = proposal
```

```
            accept_times += 1
          else:
            x = x #proposal is rejected
            reject_times += 1
        else:
          x = x #proposal is rejected
          reject_times += 1
        #Make adapatation every 50 runs satisfying the diminishing condition to assure ergodicity
        if i%50 == 0:
          acc_rate = 1 - reject_times/float(i+1)
          if acc_rate > rate_set:
            length += min(0.01, 1/float(i+1))
          elif acc_rate < rate_set:
            length -= min(0.01, 1/float(i+1))
        i +=1
      return x_1_pos,y_1_pos,time,1-(reject_times/float(step)), accept_times, reject_times



    #To estimate Var(1/n \sum g(Xi)) where g is taken as the x coordinate of the first particle
    def get_variance(step,A,B,C, rate):
      k = 1
      mean_0 = []
      while k <= 20:
        a = run(step,A,B,C, rate)
```

```
        mean_0.append(mean(a[0]))
        k += 1
    return (std(mean_0))**2


def get_corr_x0_xi(l,i):
    x=[]
    y=[]
    k = 0
    while k + i < len(l):
        x.append(l[k])
        y.append(l[k+i])
        k += 1
    return correlation(x,y)


#To get the graph of auto correlation time
def get_corr_graph(l,step):
    a = []
    b = []
    i = 1
    while i <= step:
        a.append(get_corr_x0_xi(l,i))
        b.append(i)
        i += 1
    return a,b
```

```
def get_iact(step,A,B,C,rate):
    '''Here g is taken as the first coordinate of the vector'''

    x1_final = run(step,A,B,C,rate)[0]
    sum = 0
    i = 1
    k = 0
    while i <= step and k <= 1:
        q = get_corr_x0_xi(x1_final,i)
        if q < 0:
            k += 1
        sum += q
        i += 1
    return 1 + 2*sum, i

#set the cutoff value manually
def get_iact_manual(step,A,B,C,cutoff, rate):
    '''Here g is taken as the first coordinate of the vector'''

    x1_final = run(step,A,B,C,rate)[0]
    sum = 0
    i = 1
    while i <= cutoff:
```

```
        q = get_corr_x0_xi(x1_final,i)

        sum += q

        i += 1

    return 1 + 2*sum, i


def get_withop(step,l):

    '''Here g is taken as the first coordinate of the vector'''


    x1_final = l

    sum = 0

    i = 1

    k = 0

    while i <= step and k <= 1:

        q = get_corr_x0_xi(x1_final,i)

        if q < 0:

            k += 1

        sum += q

        i += 1

    return 1 + 2*sum, i


a = run(300000,200,200,0,0.234)

taug = get_withop(50000,a[0])

var1 = std(a[0])**2

var2 = get_variance(300000,200,200,0,0.234)

print "tau g", taug[0]
```

```
print "lag", taug[1]

print "var(g(X))", var1

print "other variance", var2

print "ratio", 300000*var2/float(taug[0])



c = get_corr_graph(a[0],50000)

plt.plot(c[1],c[0], "b")

plt.show()
```

III.'''Python Code for Case5'''

'''adaptive full ma with uniform proposal'''

```
from numpy import *

from scipy import *

from math import *

from random import *

import matplotlib.pyplot as plt

from statistics import *


''' 3 point process with a specific density function on [0,1.5] * [0,1]'''


#initial position

x_1 = 1.5*random()
```

```
y_1 = random()

x_2 = 1.5*random()

y_2 = random()

x_3 = 1.5*random()

y_3 = random()


ini_pos = array([x_1,y_1,x_2,y_2,x_3,y_3],float)


#Now define pdf

def h(x, A,B,C):

   #x is the array for positions of the 3 particles

   sum_dis = float(sqrt((x[0]-x[2])**2+(x[1]-x[3])**2) + sqrt((x[0]-x[4])**2+(x[1]-x[5])**2) + sqrt((x[2]-x[4])**2+(x[3]-x[5])**2))

   sum_x = float(x[0]+x[2]+x[4])

   value = A*sum_dis + 0.04*B*(1/float(sqrt((x[0]-x[2])**2+(x[1]-x[3])**2))+1/float(sqrt((x[0]-x[4])**2+(x[1]-x[5])**2))+1/float(sqrt((x[2]-x[4])**2+(x[3]-x[5])**2))) + C*sum_x

   return value


#def pi(x,A,B,C):

   ##target distribution up to a constant

   #if 0<=x[0]<=1.5 and 0<=x[2]<=1.5 and 0<=x[4]<=1.5 and 0<=x[1]<=1 and 0<=x[3]<=1 and 0<=x[5]<=1:

      ##restricting to pink region

      #return exp(-h(x,A,B,C))

   #else:

      #return 0
```

```
#We commented out the definition of pdf because we are doing the rejection/acceptance on log scale
#to avoid numerical errors
'''Full Metropolis Adaptive Algorithm to Target Acceptance Rate with Unifrom Proposal Distribution'''

def run(step,A,B,C,rate_set):

    x = ini_pos
    reject_times = 0
    accept_times = 0

    x_1_pos = []
    y_1_pos = []
    time = []
    length = 0.1

    i = 1
    while i <= step:
        time.append(i)
        x_1_pos.append(x[0])
        y_1_pos.append(x[1])

        dummy = []
        k = 1
        while k <= 6:
            dummy.append(uniform(-length,length))
```

```
        k +=1

    increment = array(dummy)
    proposal = x + increment

    if (0<=proposal[0]<=1.5 and 0<=proposal[2]<=1.5 and 0<=proposal[4]<=1.5 and 0<=proposal[1]<=1 and 0<=proposal[3]<=1 and 0<=proposal[5]<=1):
        accept = h(x,A,B,C) - h(proposal,A,B,C)

        k = random()
        if log(k) < accept:
            x = proposal
            accept_times += 1
        else:
            x = x #proposal is rejected
            reject_times += 1
    else:
        x = x #proposal is rejected
        reject_times += 1

    #Make adapatation every 50 runs satisfying the diminishing condition to assure ergodicity
    if i%50 == 0:
        acc_rate = 1 - reject_times/float(i)
        if acc_rate > rate_set:
            length += min(0.01, 1/float(i))
        if acc_rate < rate_set:
```

```
            length -= min(0.01, 1/float(i))
        i +=1
    return x_1_pos,y_1_pos,time,1-(reject_times/float(step)), accept_times, reject_times



#To estimate Var(1/n \sum g(Xi)) where g is taken as the x coordinate of the first particle
def get_variance(step,A,B,C, rate):
    k = 1
    mean_0 = []
    while k <= 20:
        a = run(step,A,B,C, rate)
        mean_0.append(mean(a[0]))
        k += 1
    return (std(mean_0))**2


def get_corr_x0_xi(l,i):
    x=[]
    y=[]
    k = 0
    while k + i < len(l):
        x.append(l[k])
        y.append(l[k+i])
        k += 1
    return correlation(x,y)
```

```
#To get the graph of auto correlation time

def get_corr_graph(l,step):

    a = []

    b = []

    i = 1

    while i <= step:

        a.append(get_corr_x0_xi(l,i))

        b.append(i)

        i += 1

    return a,b




def get_iact(step,A,B,C,rate):

    '''Here g is taken as the first coordinate of the vector'''


    x1_final = run(step,A,B,C,rate)[0]

    sum = 0

    i = 1

    k = 0

    while i <= step and k <= 1:

        q = get_corr_x0_xi(x1_final,i)

        if q < 0:

            k += 1
```

```
        sum += q
        i += 1
    return 1 + 2*sum, i


#set the cutoff value manually
def get_iact_manual(step,A,B,C,cutoff, rate):
    '''Here g is taken as the first coordinate of the vector'''


    x1_final = run(step,A,B,C,rate)[0]
    sum = 0
    i = 1
    while i <= cutoff:
        q = get_corr_x0_xi(x1_final,i)
        sum += q
        i += 1
    return 1 + 2*sum, i


def get_withop(step,l):
    '''Here g is taken as the first coordinate of the vector'''


    x1_final = l
    sum = 0
    i = 1
    k = 0
    while i <= step and k <= 1:
```

```
        q = get_corr_x0_xi(x1_final,i)
        if q < 0:
            k += 1
        sum += q
        i += 1
    return 1 + 2*sum, i


a = run(300000,200,200,0,0.234)
taug = get_withop(50000,a[0])
var1 = std(a[0])**2
var2 = get_variance(300000,200,200,0,0.234)
print "tau g", taug[0]
print "lag", taug[1]
print "var(g(X))", var1
print "other variance", var2
print "ratio", 300000*var2/float(taug[0])



#c = get_corr_graph(a[0],50000)
#plt.plot(c[1],c[0], "b")
#plt.show()
```

'''adaptive full ma with normal proposal'''

```
from numpy import *
```

```
from scipy import *

from math import *

from random import *

import matplotlib.pyplot as plt

from statistics import *


''' 3 point process with a specific density function on [0,1.5] * [0,1]'''


#initial position

x_1 = 1.5*random()

y_1 = random()

x_2 = 1.5*random()

y_2 = random()

x_3 = 1.5*random()

y_3 = random()


ini_pos = array([x_1,y_1,x_2,y_2,x_3,y_3],float)


#Now define pdf

def h(x, A,B,C):

    #x is the array for positions of the 3 particles

    sum_dis = float(sqrt((x[0]-x[2])**2+(x[1]-x[3])**2) + sqrt((x[0]-x[4])**2+(x[1]-x[5])**2) + sqrt((x[2]-
x[4])**2+(x[3]-x[5])**2))

    sum_x = float(x[0]+x[2]+x[4])

    value = A*sum_dis + 0.04*B*(1/float(sqrt((x[0]-x[2])**2+(x[1]-x[3])**2))+1/float(sqrt((x[0]-
x[4])**2+(x[1]-x[5])**2))+1/float(sqrt((x[2]-x[4])**2+(x[3]-x[5])**2))) + C*sum_x
```

```
    return value


#def pi(x,A,B,C):
    ##target distribution up to a constant
    #if 0<=x[0]<=1.5 and 0<=x[2]<=1.5 and 0<=x[4]<=1.5 and 0<=x[1]<=1 and 0<=x[3]<=1 and 0<=x[5]<=1:
        ##restricting to pink region
        #return exp(-h(x,A,B,C))
    #else:
        #return 0


#We commented out the definition of pdf because we are doing the rejection/acceptance on log scale
#to avoid numerical errors
'''Full Metropolis Adaptive Algorithm to Target Acceptance Rate with Normal Proposal Distribution'''


def run(step,A,B,C,rate_set):


    x = ini_pos
    reject_times = 0
    accept_times = 0


    x_1_pos = []
    y_1_pos = []
    time = []
    length = 0.01
    #initial value of the scaling factor is 0.01
```

```
    i = 1
    while i <= step:
        time.append(i)
        x_1_pos.append(x[0])
        y_1_pos.append(x[1])

        dummy = []
        k = 1
        while k <= 6:
            #We have to make sure the scaling factor is greater than 0
            dummy.append(normalvariate(0,max(length,0.001)))
            k +=1

        increment = array(dummy)
        proposal = x + increment

        if (0<=proposal[0]<=1.5 and 0<=proposal[2]<=1.5 and 0<=proposal[4]<=1.5 and 0<=proposal[1]<=1
and 0<=proposal[3]<=1 and 0<=proposal[5]<=1):
            accept = h(x,A,B,C) - h(proposal,A,B,C)

            k = random()
            if log(k) < accept:
                x = proposal
                accept_times += 1
            else:
```

```
            x = x #proposal is rejected
            reject_times += 1
        else:
          x = x #proposal is rejected
          reject_times += 1

        #Make adapatation every 50 runs satisfying the diminishing condition to assure ergodicity
        #If acceptance rate is too low we shrink scaling factor and vice versa
        if i%50 == 0:
          acc_rate = 1 - reject_times/float(i)
          if acc_rate > rate_set:
            length += min(0.002, 1/float(i))
          if acc_rate < rate_set:
            length -= min(0.002, 1/float(i))
        i +=1
    return x_1_pos,y_1_pos,time,1-(reject_times/float(step)), accept_times, reject_times


#To estimate Var(1/n \sum g(Xi)) where g is taken as the x coordinate of the first particle
def get_variance(step,A,B,C, rate):
    k = 1
    mean_0 = []
    while k <= 20:
        a = run(step,A,B,C, rate)
        mean_0.append(mean(a[0]))
```

```
        k += 1
    return (std(mean_0))**2


def get_corr_x0_xi(l,i):
    x=[]
    y=[]
    k = 0
    while k + i < len(l):
        x.append(l[k])
        y.append(l[k+i])
        k += 1
    return correlation(x,y)


#To get the graph of auto correlation time
def get_corr_graph(l,step):
    a = []
    b = []
    i = 1
    while i <= step:
        a.append(get_corr_x0_xi(l,i))
        b.append(i)
        i += 1
    return a,b
```

```
def get_iact(step,A,B,C,rate):
    '''Here g is taken as the first coordinate of the vector'''

    x1_final = run(step,A,B,C,rate)[0]
    sum = 0
    i = 1
    k = 0
    while i <= step and k <= 1:
        q = get_corr_x0_xi(x1_final,i)
        if q < 0:
            k += 1
        sum += q
        i += 1
    return 1 + 2*sum, i

#set the cutoff value manually
def get_iact_manual(step,A,B,C,cutoff, rate):
    '''Here g is taken as the first coordinate of the vector'''

    x1_final = run(step,A,B,C,rate)[0]
    sum = 0
    i = 1
    while i <= cutoff:
        q = get_corr_x0_xi(x1_final,i)
```

```
        sum += q
        i += 1
    return 1 + 2*sum, i

def get_withop(step,l):
    '''Here g is taken as the first coordinate of the vector'''

    x1_final = l
    sum = 0
    i = 1
    k = 0
    while i <= step and k <= 1:
        q = get_corr_x0_xi(x1_final,i)
        if q < 0:
            k += 1
        sum += q
        i += 1
    return 1 + 2*sum, i

a = run(300000,200,200,0,0.234)
taug = get_withop(50000,a[0])
var1 = std(a[0])**2
var2 = get_variance(300000,200,200,0,0.234)
print "tau g", taug[0]
print "lag", taug[1]
```

```
print "var(g(X))", var1

print "other variance", var2

print "ratio", 300000*var2/float(taug[0])



#c = get_corr_graph(a[0],50000)

#plt.plot(c[1],c[0], "b")

#plt.show()
```

IV.'''Python Code for Case6'''

```
'''Code for Case 6 Translation adaptive algorithm with target acceptance rate and Uniform proposal distribution'''


from numpy import *

from scipy import *

from math import *

from random import *

import matplotlib.pyplot as plt

from statistics import *


''' 3 point process with a specific density function on [0,1.5] * [0,1]'''


#initial position

x_1 = 1.5*random()
```

```
y_1 = random()

x_2 = 1.5*random()

y_2 = random()

x_3 = 1.5*random()

y_3 = random()


ini_pos = array([x_1,y_1,x_2,y_2,x_3,y_3],float)


def h(x, A,B,C):

   #x is the array for positions of the 3 particles

   sum_dis = float(sqrt((x[0]-x[2])**2+(x[1]-x[3])**2) + sqrt((x[0]-x[4])**2+(x[1]-x[5])**2) + sqrt((x[2]-
x[4])**2+(x[3]-x[5])**2))

   sum_x = float(x[0]+x[2]+x[4])

   value = A*sum_dis + 0.04*B*(1/float(sqrt((x[0]-x[2])**2+(x[1]-x[3])**2))+1/float(sqrt((x[0]-
x[4])**2+(x[1]-x[5])**2))+1/float(sqrt((x[2]-x[4])**2+(x[3]-x[5])**2))) + C*sum_x

   return value




def run(step,A,B,C,rate_set):


   x = ini_pos

   reject_times = 0

   accept_times = 0


   x_1_pos = []

   y_1_pos = []
```

```
time = []
length = 0.1

algorithm = 1
#algorithm controls what algorithm to use

i = 1
while i <= step:
    time.append(i)
    x_1_pos.append(x[0])
    y_1_pos.append(x[1])

    if algorithm == 1:
        #if we use the usual Metropolis within Gibbs algorithm
        #pick a point at random
        dummy = [0,0,0,0,0,0]
        a = randint(1,3)
        r1 = uniform(-length,length)
        r2 = uniform(-length,length)
        x_pos = a*2-2
        dummy[x_pos] = r1
        dummy[x_pos+1] = r2

        increment = array(dummy)
        proposal = x + increment
```

```
        if (0<=proposal[0]<=1.5 and 0<=proposal[2]<=1.5 and 0<=proposal[4]<=1.5 and 
0<=proposal[1]<=1 and 0<=proposal[3]<=1 and 0<=proposal[5]<=1):
            accept = h(x,A,B,C) - h(proposal,A,B,C)

            k = random()
            if log(k) < accept:
                x = proposal
                accept_times += 1
            else:
                x = x #proposal is rejected
                reject_times += 1
        else:
            x = x #proposal is rejected
            reject_times += 1

        if i%50 == 0:
            #Make adapatation every 50 runs satisfying the diminishing condition to assure ergodicity
            acc_rate = 1 - reject_times/float(i)
            if acc_rate > rate_set:
                length += min(0.01, 1/float(i))
            if acc_rate < rate_set:
                length -= min(0.01, 1/float(i))

            #we want to change algorithm with probability p(i)=1/(i^(1/8)), this satisfies diminishing 
condition
```

```
            #It turns out the diminishing condition for algorithm swtiching is not necessary
            #to assure ergodicity in this case, but the simulation results turn out to be surprisingly
            #good. The code was not changed. However in the modified Java Applet we got rid of the
            #diminishing condition for algorithm switching.
            pro_to_change = 1/float(i**(1/8))
            k2 = random()
            if k2 < pro_to_change:
                algorithm = 2
        i += 1
    elif algorithm == 2:
        #if we use the translation algorithm
        r1 = uniform(-0.4,0.4)
        r2 = uniform(-0.4,0.4)
        dummy = [r1,r2,r1,r2,r1,r2]
        increment = array(dummy)
        proposal = x + increment

        if (0<=proposal[0]<=1.5 and 0<=proposal[2]<=1.5 and 0<=proposal[4]<=1.5 and
0<=proposal[1]<=1 and 0<=proposal[3]<=1 and 0<=proposal[5]<=1):
            accept = h(x,A,B,C) - h(proposal,A,B,C)

            k = random()
            if log(k) < accept:
                x = proposal
                accept_times += 1
            else:
```

```
                x = x #proposal is rejected

                reject_times += 1

            else:

              x = x #proposal is rejected if the proposal is not in the region

              reject_times += 1


            algorithm = 1

            i += 1

    return x_1_pos,y_1_pos,time,1-(reject_times/float(step)), accept_times, reject_times



#To estimate Var(1/n \sum g(Xi)) where g is taken as the x coordinate of the first particle

def get_variance(step,A,B,C, rate):

    k = 1

    mean_0 = []

    while k <= 20:

        a = run(step,A,B,C, rate)

        mean_0.append(mean(a[0]))

        k += 1

    return (std(mean_0))**2


def get_corr_x0_xi(l,i):

    x=[]

    y=[]

    k = 0
```

```
    while k + i < len(l):
        x.append(l[k])
        y.append(l[k+i])
        k += 1
    return correlation(x,y)

#To get the graph of auto correlation time
def get_corr_graph(l,step):
    a = []
    b = []
    i = 1
    while i <= step:
        a.append(get_corr_x0_xi(l,i))
        b.append(i)
        i += 1
    return a,b



def get_iact(step,A,B,C,rate):
    '''Here g is taken as the first coordinate of the vector'''

    x1_final = run(step,A,B,C,rate)[0]
    sum = 0
    i = 1
```

```
    k = 0
    while i <= step and k <= 1:
        q = get_corr_x0_xi(x1_final,i)
        if q < 0:
            k += 1
        sum += q
        i += 1
    return 1 + 2*sum, i


#set the cutoff value manually
def get_iact_manual(step,A,B,C,cutoff, rate):
    '''Here g is taken as the first coordinate of the vector'''


    x1_final = run(step,A,B,C,rate)[0]
    sum = 0
    i = 1
    while i <= cutoff:
        q = get_corr_x0_xi(x1_final,i)
        sum += q
        i += 1
    return 1 + 2*sum, i


def get_withop(step,l):
    '''Here g is taken as the first coordinate of the vector'''
```

```
    x1_final = l

    sum = 0

    i = 1

    k = 0

    while i <= step and k <= 1:

        q = get_corr_x0_xi(x1_final,i)

        if q < 0:

            k += 1

        sum += q

        i += 1

    return 1 + 2*sum, i


a = run(300000,200,200,0,0.234)

taug = get_withop(50000,a[0])

var1 = std(a[0])**2

var2 = get_variance(300000,200,200,0,0.234)

print "tau g", taug[0]

print "lag", taug[1]

print "var(g(X))", var1

print "other variance", var2

print "ratio", 300000*var2/float(taug[0])




#c = get_corr_graph(a[0],50000)

#plt.plot(c[1],c[0], "b")
```

#plt.show()

V.'''Modified Java Code for Point Process Applet with Adaptive MCMC Algorithms'''

*Originally created by Jeffrey Rosenthal

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package kai;


import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.Button;
import java.awt.Checkbox;
import java.awt.Color;
import java.awt.Event;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Panel;
import java.awt.TextField;
import java.util.Random;
import javax.swing.JOptionPane;
```

```
/**
 *
 * @author 08ekanay, 09yangka
 */
public class KaiApp extends Applet implements Runnable {

    int testrandomnumber = 0;

    int i, j, k;
    double SUM, mean, tmpdouble;
    int iteration = 0;
    int numacc = 0;
    int MAXNUMPARTS = 20;
    int MINNUMPARTS = 2;
    int NUMPARTS = 5;
    int PRAD = 10;
    int xmin = PRAD;
    int ymin = PRAD;
    int xmax;
    int ymax;
    int textwidth = 200;
    int textx;
    int texty = 30;
    int textinc = 30;
```

```
int chosen = 1;

int proprad = 50;

//int xprop, yprop;

double sumdif;

double SCALE = 400.0;

int A = 5;

int B = 5;

int C = 0;

int parincr = 1;

double sumdist, meandist;

boolean REDRAW = true;

boolean FORCEDREDRAW = true;

int speedcontrol = 2;

int MINNUMSTATES = 3;

int MAXNUMSTATES = 10;

double idens[] = new double[MAXNUMSTATES + 1];

int denscount[] = new int[MAXNUMSTATES + 1];

int xcoord[] = new int[MAXNUMPARTS + 1];

int ycoord[] = new int[MAXNUMPARTS + 1];

int xprop[] = new int[MAXNUMPARTS + 1];

int yprop[] = new int[MAXNUMPARTS + 1];

Random randgen = new Random();

// Color panelcol = Color.gray;

Color panelcol = new Color(178, 178, 178);

int segment, itmp;
```

```
boolean accept, JUMPONE, RECOUNT;

boolean keyjustpressed = false;

boolean RESTARTING = true;

int ADAPTLEVEL = 1;

Image holdImage;

Graphics holdGraphics;

Font strongfont = new Font("TimesRoman", Font.BOLD, 18);


Checkbox bKai1 = new Checkbox("Target Acceptance");

Checkbox bKai2 = new Checkbox("With Translation Algorithm");


private double curaccrate;

private double inputval;

private boolean adaptationOn;

private boolean translationOn;

double transRand = Math.random();

private int numtrans = 0;

public void init() {

   this.setSize(800, 400);

   holdImage = createImage(size().width, size().height);

   holdGraphics = holdImage.getGraphics();

   xmax = size().width - textwidth - PRAD;

   ymax = size().height - PRAD;

   textx = xmax + 50;

   // repaint();
```

```
        this.setLayout(new BorderLayout() );

        Panel kaiP = new Panel(new BorderLayout());

        //Button bKai = new Button("Addaptation");

        kaiP.add(bKai1, BorderLayout.NORTH);

        kaiP.add(bKai2, BorderLayout.SOUTH);

        //TextField lKai = new TextField(6);

        //kaiP.add(lKai, BorderLayout.SOUTH);

        this.add(kaiP, BorderLayout.EAST);

        //JOptionPane.showInputDialog("gdfgdfg");


    }


    public void standardAlgo(){


//  System.out.println("Here: xmin=" + xmin + ";   xmax=" + xmax);


            if (RESTARTING) {

                for (i = 1; i <= NUMPARTS; i++) {

                    xcoord[i] = xmin + randgen.nextInt(xmax - xmin + 1);

                    ycoord[i] = ymin + randgen.nextInt(ymax - ymin + 1);

                }

                zerocount();

                JUMPONE = RECOUNT = false;

                REDRAW = true;

                RESTARTING = false;
```

```
        } else if (RECOUNT) {
          zerocount();
          RECOUNT = false;
        }

// System.out.println("Here");

        // Start fresh iteration.

// System.out.println("Here");

        // Compute the proposal and accept/reject.
        chosen = 1 + randgen.nextInt(NUMPARTS);
        int xinc = randgen.nextInt(2 * proprad + 1) - proprad;
        int yinc = randgen.nextInt(2 * proprad + 1) - proprad;
        xprop[chosen] = xcoord[chosen] + xinc;
        yprop[chosen] = ycoord[chosen] + yinc;

        accept = false;

        if ((xprop[chosen] >= xmin) && (xprop[chosen] <= xmax)
              && (yprop[chosen] >= ymin) && (yprop[chosen] <= ymax)) {

          sumdif = V(xprop[chosen], yprop[chosen]) - V(xcoord[chosen], ycoord[chosen]);
          for (i = 1; i <= NUMPARTS; i++) {
```

```
                if (i != chosen) {
                    sumdif = sumdif + h(xcoord[i], ycoord[i], xprop[chosen], yprop[chosen])
                            - h(xcoord[i], ycoord[i], xcoord[chosen], ycoord[chosen]);
                }
            }

            if (randgen.nextDouble() < Math.exp(-sumdif)) {
                accept = true;
            }

        }

// System.out.println("Here");

        // Display the various segments of the iteration.
        for (segment = 1; segment <= 4; segment++) {
            updatenpause();
        }

// System.out.println("Here");

        // Update state.
        if (accept) {
            xcoord[chosen] = xprop[chosen];
            ycoord[chosen] = yprop[chosen];
```

```
        numacc++;
      }

      // Update functional values.
      sumdist = 0;
      for (i = 1; i < NUMPARTS; i++) {
        for (j = i + 1; j <= NUMPARTS; j++) {
          sumdist = sumdist + dist(xcoord[i], ycoord[i], xcoord[j], ycoord[j]);
        }
      }
      meandist = sumdist / (NUMPARTS * (NUMPARTS - 1) / 2);
      SUM = SUM + meandist;
      iteration++;

      //calculating the current acceptance rate
      curaccrate = Math.round(10000.0 * numacc / iteration) / 100.0;



}

public void adaptation(){

        if (iteration % 1 == 0) {
          double random = Math.random();
          if (random < 1 / Math.log(iteration)) {
```

```
            if (curaccrate < inputval) {
                if (proprad > 1)
                    proprad--;
            }else if (curaccrate > inputval){
                proprad++;
            }
        }

    }

}

public void translationAlgo() {
    if (RESTARTING) {
        for (i = 1; i <= NUMPARTS; i++) {
            xcoord[i] = xmin + randgen.nextInt(xmax - xmin + 1);
            ycoord[i] = ymin + randgen.nextInt(ymax - ymin + 1);
        }
        zerocount();
        JUMPONE = RECOUNT = false;
        REDRAW = true;
        RESTARTING = false;
    } else if (RECOUNT) {
        zerocount();
        RECOUNT = false;
```

```
        }

        int xinc = randgen.nextInt(2 * proprad + 1) - proprad;
        int yinc = randgen.nextInt(2 * proprad + 1) - proprad;
        accept = true;
        for (int i = 1; i <= NUMPARTS; i++) {
            xprop[i] = xcoord[i] + xinc;
            yprop[i] = ycoord[i] + yinc;
            if (!((xprop[i] >= xmin) && (xprop[i] <= xmax)
                    && (yprop[i] >= ymin) && (yprop[i] <= ymax))) {
                accept = false;

            }
        }
//      for(int i=1; i<NUMPARTS; i++){
//          for (segment = 1; segment <= 4; segment++) {
//              updatenpause();
//          }
//      }
        for (segment = 1; segment <= 4; segment++) {
                updatenpause();
        }
        if (accept) {
            numacc++;
```

```
    for (int i = 1; i <= NUMPARTS; i++) {
        chosen = i;

        xcoord[i] = xcoord[i] + xinc;
        ycoord[i] = ycoord[i] + yinc;

        //System.out.println("Accepted xcoord of " + iteration + " " + xcoord[i]);


    }

}
sumdist = 0;
for (int k = 1; k < NUMPARTS; k++) {
    for (j = k + 1; j <= NUMPARTS; j++) {
        sumdist = sumdist + dist(xcoord[k], ycoord[k], xcoord[j], ycoord[j]);
    }
}
meandist = sumdist / (NUMPARTS * (NUMPARTS - 1) / 2);
SUM = SUM + meandist;


curaccrate = Math.round(10000.0 * numacc / iteration) / 100.0;
//System.out.println("Accepted numacc and iter of " + numacc + " " + iteration);
iteration++;
```

```
}

public void run() {

    while (true) {
      if (translationOn) {
          transRand = Math.random();
          if (transRand < 0.5){
           this.translationAlgo();
           numtrans++;
           System.out.println("Numtrans: " + numtrans);
          }else{
            this.standardAlgo();
            //System.out.println("standard");
          }
       }else{
           this.standardAlgo();
       }
      if (adaptationOn){
           this.adaptation();
       }
    }
}
```

```
public void updatenpause() {

  if ((!RESTARTING) && (!RECOUNT)
      && ((speedcontrol < 5) || (segment == 1) || keyjustpressed)
      && ((segment < 5) || accept)) {

    itmp = 10 * (speedcontrol - 5) * (speedcontrol - 5);
    if ((speedcontrol <= 5) || keyjustpressed
        || (iteration == (iteration / itmp) * itmp)) {
      repaint();
    }
    keyjustpressed = false;

    if (speedcontrol <= 1) {
      JUMPONE = false;
      while ((speedcontrol <= 1) && (JUMPONE == false)
          && (!RESTARTING) && (!RECOUNT)) {
        dosleep(50);
      }
    } else if (speedcontrol == 2) {
      dosleep(1000);
    } else if (speedcontrol == 3) {
      dosleep(250);
    } else if (speedcontrol == 4) {
      dosleep(50);
```

```
        } else if (speedcontrol == 5) {
          dosleep(40);
        } else if (speedcontrol >= 6) {
          dosleep(20);
        }

        REDRAW = FORCEDREDRAW;
        JUMPONE = false;

    }

}

public void paint(Graphics g) {

    if (REDRAW) {

        // setFont(strongfont);

        // Background rectangles.
        g.setColor(Color.pink);
        g.fillRect(0, 0, size().width - textwidth, size().height);
        g.setColor(panelcol);
        g.fillRect(size().width - textwidth, 0, textwidth, size().height);
```

```
} // end of if(REDRAW)



testrandomnumber++;

// Text matter.

g.setColor(Color.BLUE);

g.drawString("Original algorithm and Code by Jeffrey S. Rosenthal",

                textx, texty + 14 * textinc);


g.drawString("Translation Algorithm by Kai Yang "

    , textx, texty + 15 * textinc);

 g.drawString("Modified by " +

    "Chaveen Ekanayake and " +

    "Ahimsian Shanmugalingam", textx, texty + 16 * textinc);

g.setColor(Color.black);

g.drawString("Iterations: " + iteration, textx, texty + textinc);


g.drawString("A = " + A, textx, texty + 2 * textinc);

g.drawString("B = " + B, textx, texty + 3 * textinc);

g.drawString("C = " + C, textx, texty + 4 * textinc);

//g.drawString("accept = " + Math.round(10000.0 * numacc / iteration) / 100.0

   //     + "%", textx, texty + 5 * textinc);

g.drawString("accept = " + curaccrate

        + "%", textx, texty + 5 * textinc);

g.drawString("# particles = " + NUMPARTS, textx, texty + 6 * textinc);
```

```
g.drawString("speed = " + speedcontrol, textx, texty + 7 * textinc);
g.drawString("curavdist = " + Math.round(1000.0 * meandist) / 1000.0,
        textx, texty + 10 * textinc);
if (iteration > 0) {
    mean = SUM / ((double) iteration);
} else {
    mean = 0.0;
}
g.drawString("meanavdist = " + Math.round(1000.0 * mean) / 1000.0,
        textx, texty + 11 * textinc);
g.drawString("number of translations: "+ numtrans,
        textx, texty + 12 * textinc);

// Draw current state.
g.setColor(Color.black);
for (i = 1; i <= NUMPARTS; i++) {
    fillcircle(g, xcoord[i], ycoord[i], PRAD);
}

if (segment >= 2) {

    // Draw chosen point.
    if (!(translationOn && transRand < 0.5)) {
        g.setColor(Color.white);
        fillcircle(g, xcoord[chosen], ycoord[chosen], PRAD);
```

```
    }
  }

  if (segment >= 3) {

    // Draw proposal dot.
    if (segment == 3) {
      g.setColor(Color.yellow);
    } else if (accept) {
      g.setColor(Color.green);
    } else {
      g.setColor(Color.red);
    }
    if (translationOn && transRand < 0.5){
      for (int i = 1; i <= NUMPARTS; i++) {
        fillcircle(g, xprop[i], yprop[i], PRAD);
      }
    }else{
       fillcircle(g, xprop[chosen], yprop[chosen], PRAD);
    }
  }
```

```
}

public void zerocount() {

    SUM = mean = 0.0;
    iteration = numacc = 0;

}

public double h(int x1, int y1, int x2, int y2) {
    double thedist = dist(x1, y1, x2, y2);
    return (1.0 * A * thedist + 0.04 * B / thedist);
}

public double V(int x1, int y1) {
    return (2.0 * C * x1 / SCALE);
}

public double dist(int x1, int y1, int x2, int y2) {
    return (Math.sqrt(sq((x2 - x1) / SCALE) + sq((y2 - y1) / SCALE)));
}

public double sq(double rr) {
    return (rr * rr);
}
```

```
public boolean action(Event e, Object o) {
  if (e.target.equals(bKai1)){
    if (bKai1.getState()) {
      String input = JOptionPane.showInputDialog("Input target " +
          "acceptance rate e.g. 38.54");
      try {
        if (input!= null){
          inputval = Double.parseDouble(input);
          adaptationOn = true;

        }
      } catch (NumberFormatException exp) {
        JOptionPane.showMessageDialog(null, "Wrong input");
      }
    }else{
      adaptationOn = false;

    }
  }
  if (e.target.equals(bKai2)){
    if (bKai2.getState()){
      translationOn = true;

    }else{
```

```
            translationOn = false;
        }
    }
    return true;
}

public void update(Graphics g) {
    paint(holdGraphics);
    g.drawImage(holdImage, 0, 0, this);
}

public void dosleep(long nummilisecs) {
    try {
        Thread.currentThread().sleep(nummilisecs);
    } catch (InterruptedException e) {
    }
}

public void fillcircle(Graphics g, int xx, int yy, int rr) {
    g.fillOval(xx - rr, yy - rr, 2 * rr, 2 * rr);
}

public boolean keyDown(Event evt, int key) {
    char keystroke = (char) key;
```

```
if ((keystroke >= '0') && (keystroke <= '9')) {
    speedcontrol = keystroke - '0';
    if (keystroke > '0') {
        JUMPONE = true;
    }
}

if ((keystroke == 'r') || (keystroke == 'R')) {  // restart
    RESTARTING = true;
}

if ((keystroke == 'z') || (keystroke == 'Z')) {  // zero the counts
    zerocount();
}

if ((keystroke == '+') && (NUMPARTS < MAXNUMPARTS)) {
    // Increment the number of particles.
    NUMPARTS++;
    RESTARTING = true;
}

if ((keystroke == '-') && (NUMPARTS > MINNUMPARTS)) {
    // Decrement the number of particles.
    NUMPARTS--;
    RESTARTING = true;
```

```
}

if (keystroke == 'A') {
   A = A + parincr;
}

if (keystroke == 'a') {
   A = A - parincr;
}

if (keystroke == 'B') {
   B = B + parincr;
}

if (keystroke == 'b') {
   B = B - parincr;
}

if (keystroke == 'C') {
   C = C + parincr;
}

if (keystroke == 'c') {
   C = C - parincr;
}
```

```
        keyjustpressed = true;

        repaint();

        return true;

    }

    Thread t;


    public void start() {

        t = new Thread(this);

        t.start();

    }


    public void stop() {

        t.stop();

        t = null;

    }

//TODO overwrite start(), stop() and destroy() methods

}
```