UNIVERSITY OF CALIFORNIA, LOS ANGELES

STATS 202A Final Homework

UID : 704557096

Salil S. Kanetkar

December 10, 2015

Solution Problem 1:
CPP Code:

```
#include <RcppArmadilloExtensions/sample.h>
//[[Rcpp::depends(RcppArmadillo)]]
using namespace Rcpp;
using namespace arma;

// [[Rcpp::export]]
arma::vec runifC(double seed, int n)
{
  double x = seed;
  arma::vec u(n);
  for(int i = 0; i < n; ++i)
  {
    x = fmod(pow(7,5) * x, pow(2,31) - 1.0);
    u[i] = x / (pow(2,31) - 1.0);
  }
  return(u);
}

// [[Rcpp::export]]
arma::vec rnormC(double seed, int n)
{
  double pi = 3.14159265358979323846264338328O;
  double theta;
  double r;
  arma::vec u(2);
  arma::vec v(2 * n);
  for(int i = 0; i < n; ++i)
  {
    u = runifC(seed * (i + 1), 2);
    theta = 2.0 * pi * u[0];
```

```
    r = sqrt(-2 * log(u[1]));
    v[2 * i] = r * cos(theta);
    v[(2 * i) + 1] = r * sin(theta);
  }
  return(v);
}
```

R Code:

```
#For Uniform Generator
library(Rcpp)
sourceCpp("stats_final.cpp")
par(mfrow=c(1,2))
hist(runifC(12345, 10000), xlab ="U",prob=T,main="Histogram of U")
plot(runifC(12345, 10000)[seq(2,1000,by=2)], runifC(12345, 10000)[seq(3,1001,by=2)] , xlab

#For Normal Generator
hist(rnormC(76543, 10000),xlab="X and Y",main = "Histogram of X and Y", xlim =c(-4,4))
normal = rnormC(76543,10000)
x = normal[seq(1,20000,by=2)]
y = normal[seq(2,20000,by=2)]
par(mfrow =c(1,2))
hist(x , breaks =30, prob =T, xlim =c(-4,4) , xlab ="X", main = "Histogram of X")
curve(dnorm(x,mean =0, sd =1) ,lwd =2, add =T, col="blue")
hist(y , breaks =30, prob =T, xlim =c(-4,4), xlab ="Y", main = "Histogram of Y")
curve(dnorm(x,mean =0, sd =1) ,lwd =2, add =T, col="blue")
```

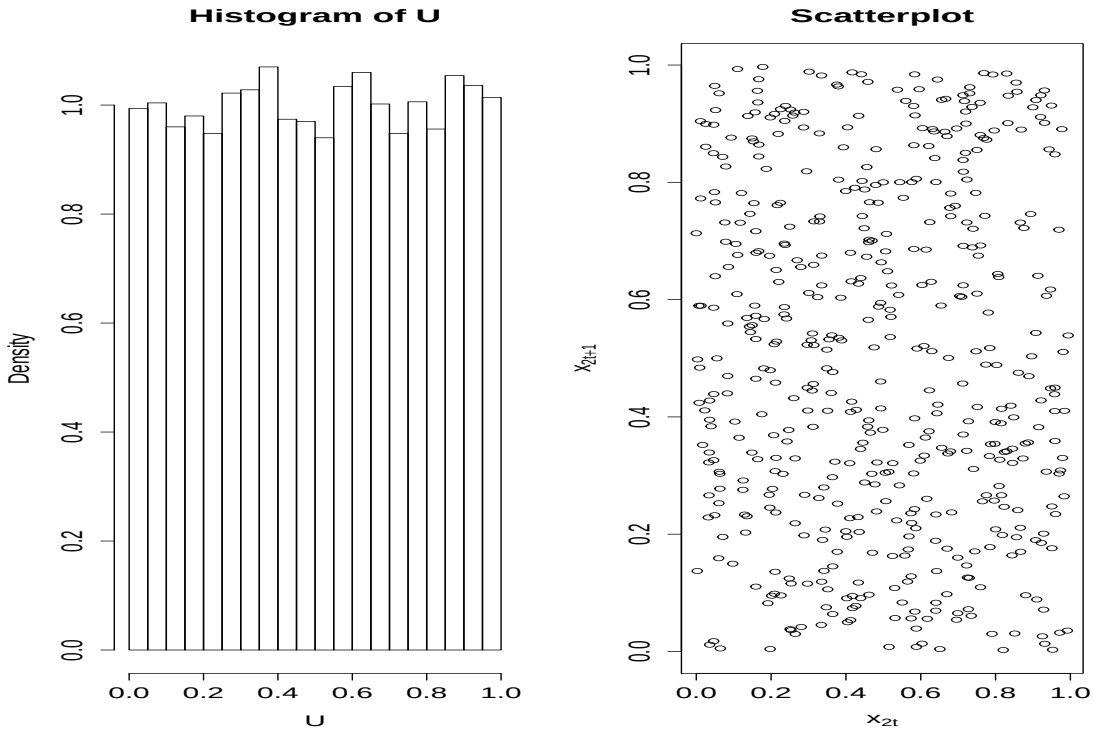

Figure 0.1: Problem 1 - Uniform Distribution using Linear Congruential Method

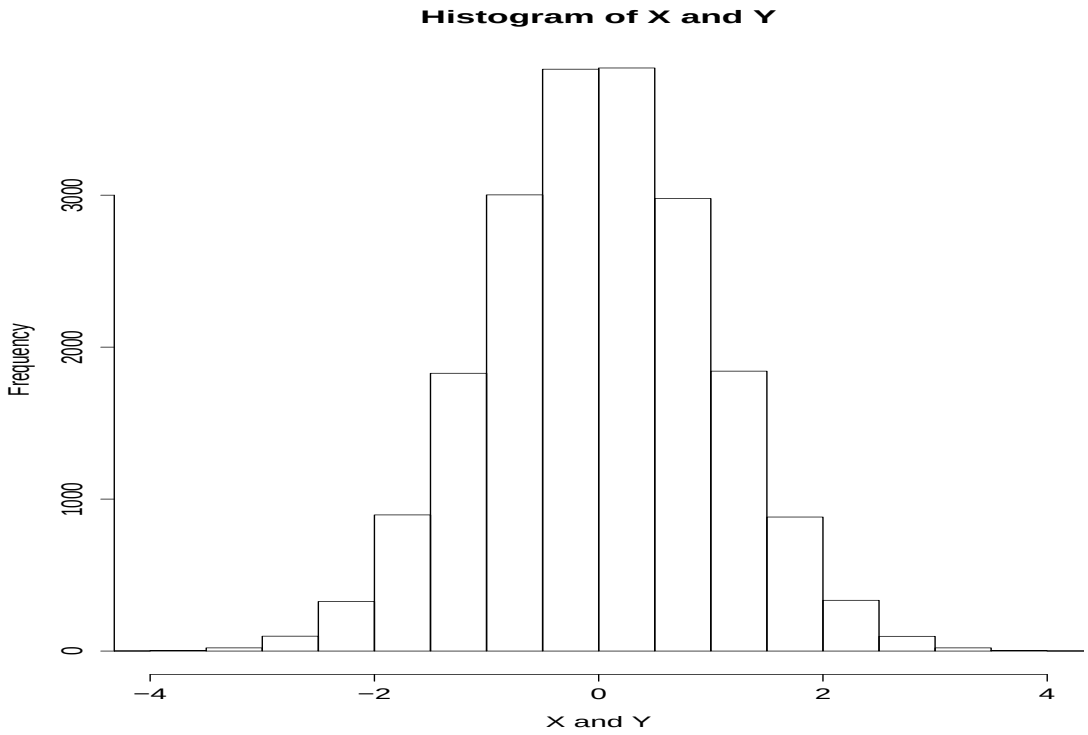

Figure 0.2: Problem 1 - Normal Distribution using Polar Transformation of X and Y

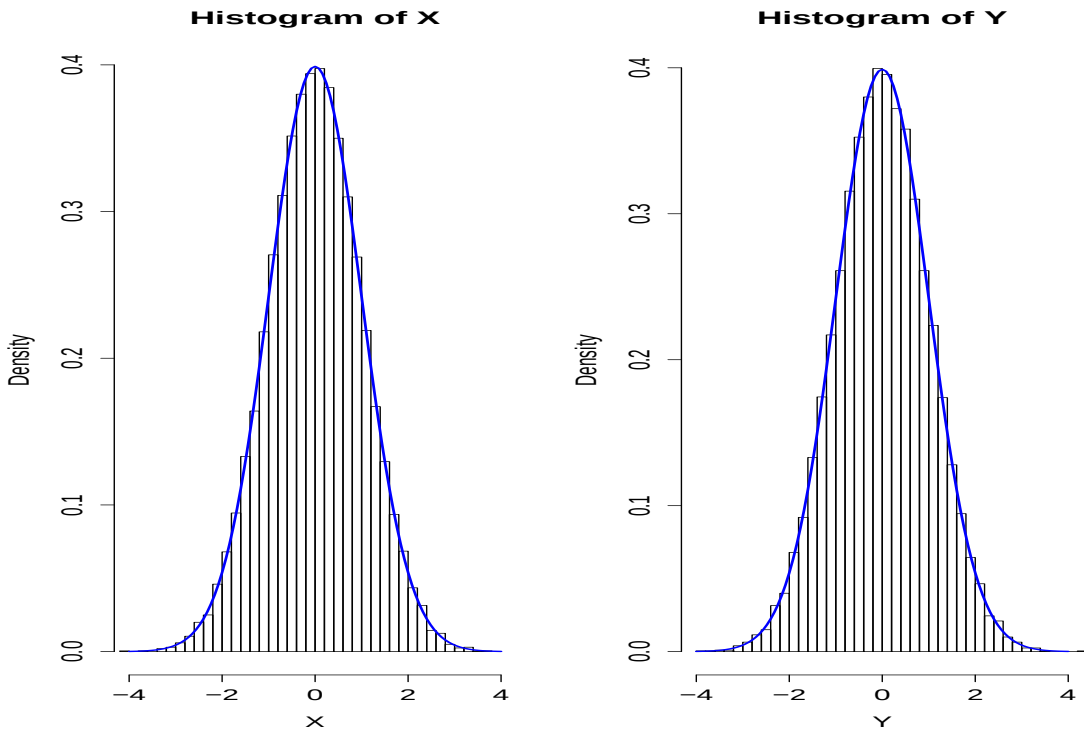

Figure 0.3: Problem 1 - Normal Distribution using Polar Transformation

Solution Problem 2:
CPP Code:

```
#include <RcppArmadillo.h>
#include <RcppArmadilloExtensions/sample.h>
//[[Rcpp::depends(RcppArmadillo)]]
using namespace Rcpp;
using namespace arma;
// [[Rcpp::export]]
double f(long x, int variance)
{
  return (exp(-pow(x,2)/(2*variance)));
}

// [[Rcpp::export]]
double alpha(long x, long y, int variance = 2)
{
  return (1.0<(f(y,variance)/f(x,variance))) ? 1.0 : (f(y,variance)/f(x,variance));
}

// [[Rcpp::export]]
arma::vec metropolisCPP(arma::vec ch, long N, long T)
{
  //arma::mat ch = clone(chain);
  long y;
  for(int n=0;n<N;n++)
  {
    long x=0;
    for(int i=0;i<T;i++)
    {
      arma::vec u1(10001);
      u1 = runifC(12345,10001);
      if(u1(n)>0.5)
      {
        y = x + 1;
      }
      else
      {
        y = x - 1;
      }
      arma::vec u2 = runifC(123,10001);
      int accept = (u2[n]<alpha(x,y));
      if(accept == 1)
      {
        x = y;
      }
      else
      {
        x = x;
      }
    }
    ch[n] = x;
```

```
  }
  return ch;
}
```

R Code:

```
library(Rcpp)
sourceCpp("stats_final.cpp")
N=10^3
T = 10^2
chain = matrix(0,1,N)
chain1 =  metropolisCPP(chain, N, T)
hist(chain1,xlim=range(c(-25,25)),freq=FALSE)
```

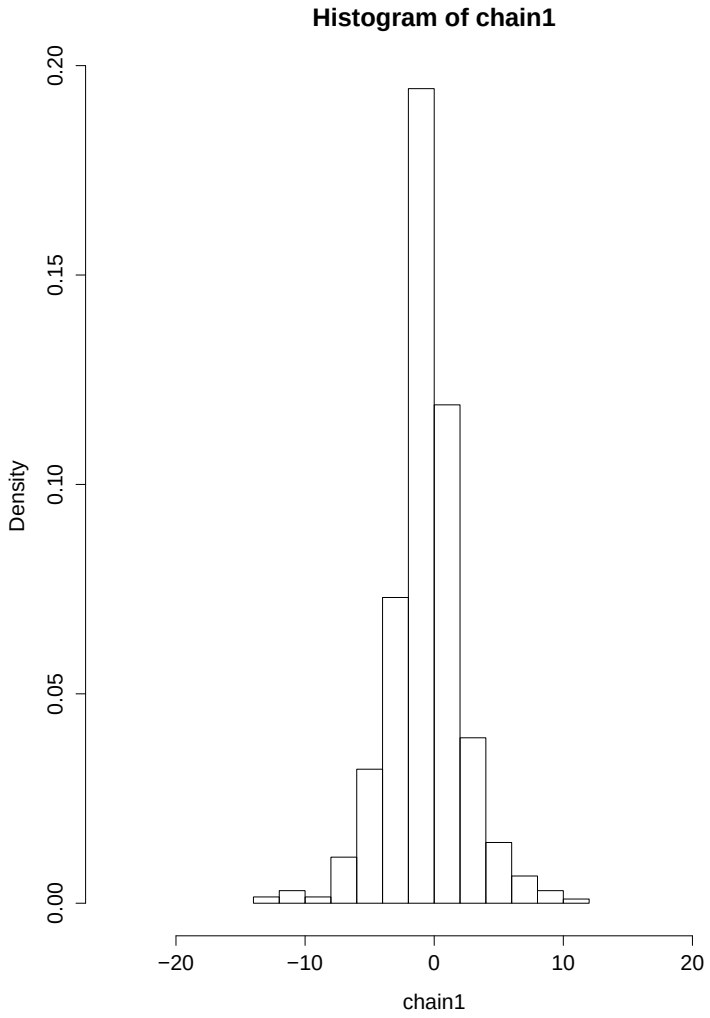

Figure 0.4: Problem 2 - Metropolis algorithm to sample from π

Solution Problem 3:

CPP Code:

```
#include <RcppArmadillo.h>
#include <RcppArmadilloExtensions/sample.h>
//[[Rcpp::depends(RcppArmadillo)]]
using namespace Rcpp;
using namespace arma;
// [[Rcpp::export]]
arma::cube gibbsCPP(NumericVector p2, int T, int M, double rho, double x0, double y0)
{
  IntegerVector dim_p2=p2.attr("dim");
  arma::cube p(p2.begin(), dim_p2[0], dim_p2[1], dim_p2[2]);
  double x, y;
  arma::vec n1 = rnormC(76543,M*T);
  //arma::vec n2 = rnormC(76543,M*T);
  for(int m=0;m<M;m++)
  {
    x = x0;
    y = y0;
    p(0,0,m) = x;
    p(1,0,m) = y;
    for(int t=1;t<T;t++)
    {
      x = n1[m*T];
      x = x * sqrt(1-pow(rho,2)) + rho * y;
      y = n1[m*T+1];
      y = y * sqrt(1-pow(rho,2)) + rho * x;
      p(0,t,m) = x;
      p(1,t,m) = y;
    }
  }
  return(p);
}
```

R Code:

```
library(Rcpp)
sourceCpp("stats_final.cpp")
gibbs<-function (T=1000, M=100, rho, x0, y0)
{
  p<-rep(0,2*M*T) #Allocate memory for results
  dim(p)<-c(2,T,M)
  p1 = gibbsCPP(p,T=1000,M=100,rho,x0,y0)
  p1
}

#making a movie
library(animation)
rho <- 0.99
M=100
par(mar=c(2,2,1,2), mfrow=c(3,3))
bvn <- gibbs(x0=-5,y0=0,M=M,rho=rho)
```

```
lims <- 8*c(-1,1)
for (t in 1:9){ plot(bvn[1,t,],bvn[2,t,],
                     xlim=lims, ylim=lims,
                     col=1:M,
                     pch=16, main=paste('t =',t))
  ani.pause(.2)
}#saving as GIF file
saveGIF({ for (t in 1:9){ plot(bvn[1,t,],bvn[2,t,], xlim = lims, ylim = lims, col=1:M,
                               pch =16, main = paste('t =',t))
}
}, movie.name =paste("bvn_gibbs_rho_", rho))
```

Below are 3 figures obtained by varying the value of ρ, X_0, Y_0

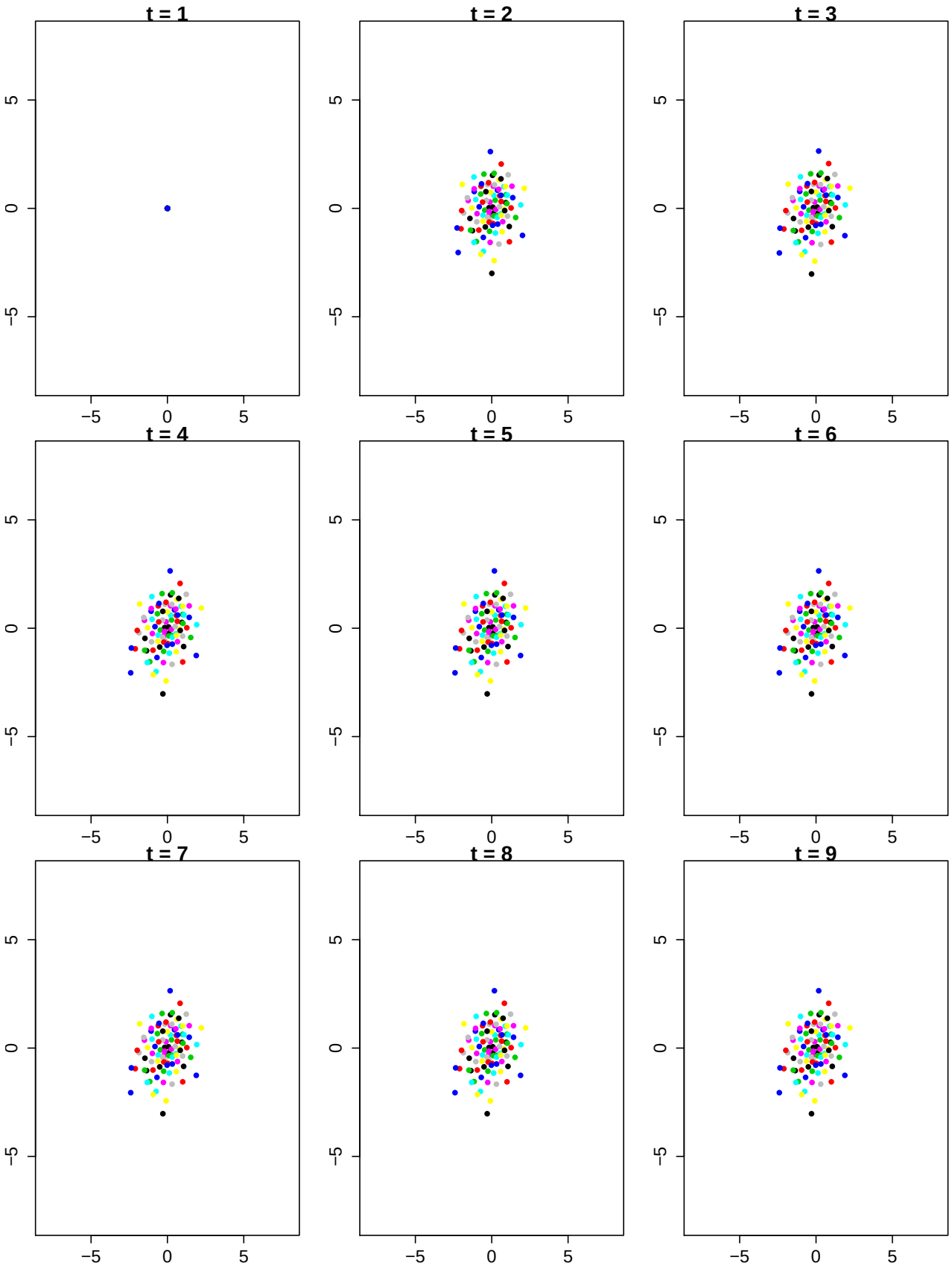

Figure 0.5: Problem 3 - $X_0 = 0, Y_0 = 0, \rho = 0.1$

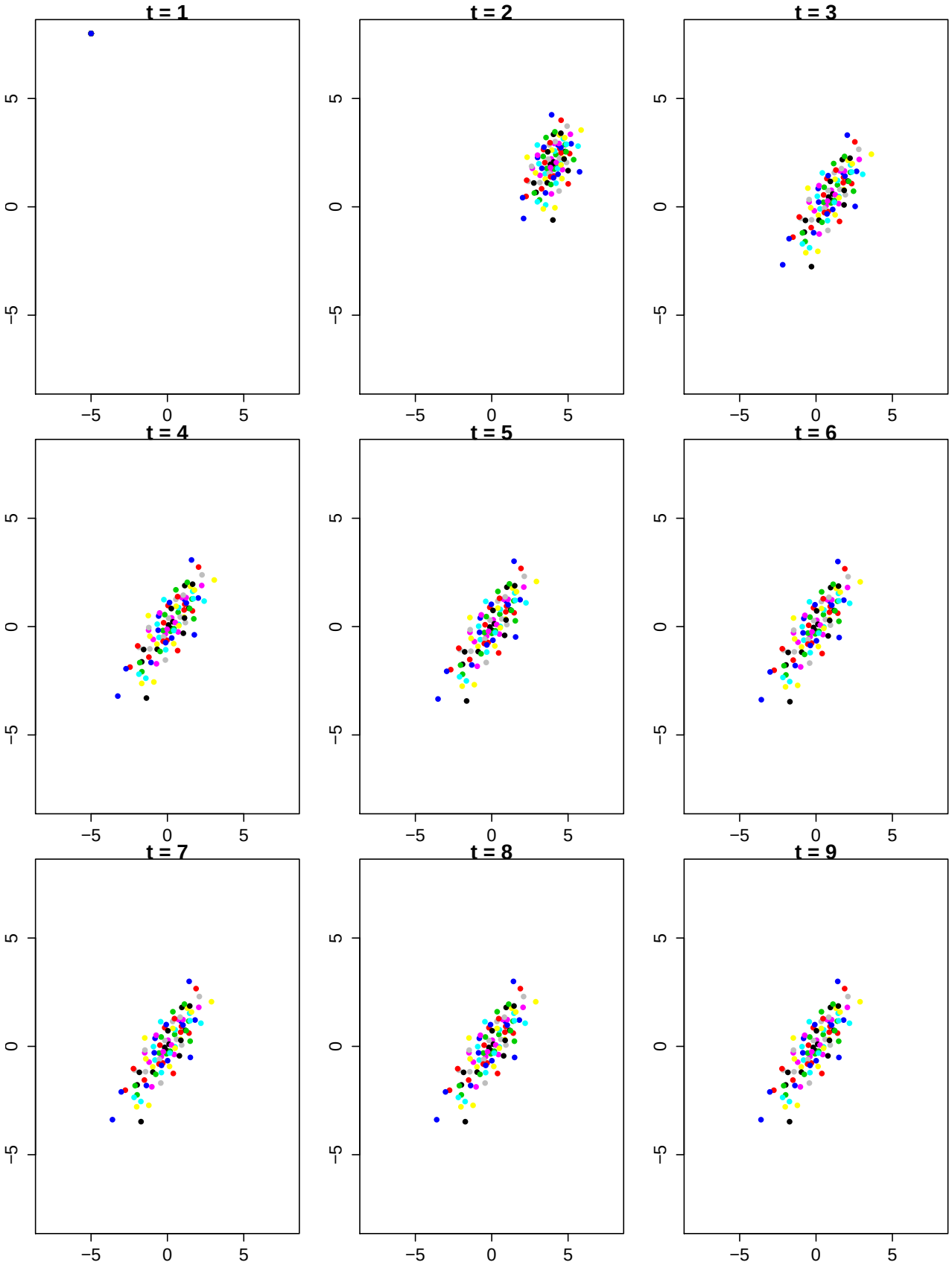

Figure 0.6: Problem 3 - $X_0 = -5, Y_0 = 8, \rho = 0.5$

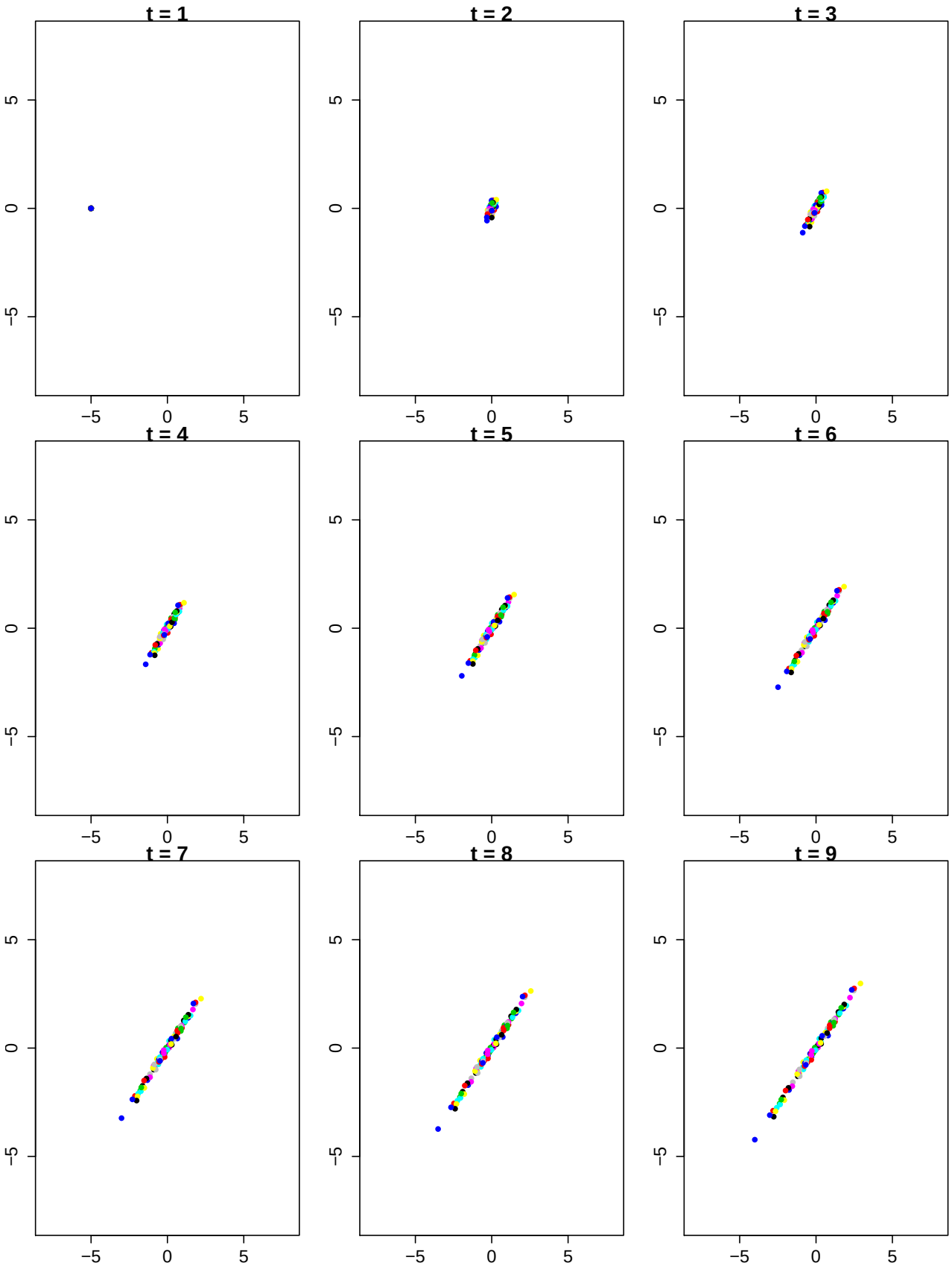

Figure 0.7: Problem 3 - $X_0 = 5, Y_0 = 0, \rho = 0.99$