MCMC Demonstrations in `MATLAB, Python`, and `R`

Paul Miles*[a], Isaac Michaud[b], and Ralph Smith[a]

[a]Department of Mathematics, North Carolina State University, Raleigh, NC 27607
[b]Department of Statistics, North Carolina State University, Raleigh, NC 27607

March 29, 2018

Contents

1 Preamble

This document is intended to present how to solve the same problem using Markov Chain Monte Carlo (MCMC) methods in three different programming languages - `Matlab`, `Python`, and `R`. It is not intended to be a user's guide on how to use each type of language, but rather a demonstration of how you can perform similar analyses using whatever language you prefer. For the purpose of this demonstration we will consider the banana example discussed in Section 2. Four different MCMC algorithms are tested in each language:

1. MH - Metropolis Hastings
2. AM - Adaptive Metropolis
3. DR - Delayed Rejection
4. DRAM - Delayed Rejection Adaptive Metropolis

A visual comparison is provided in Figure 1 presenting the pairwise correlation between the two model parameters for each language and each algorithm. The banana shaped nature of this correlation is what motivates the name of our example. Also, a comparison of the acceptance rate for each algorithm is presented in Table 1. A more detailed description of these algorithms can be found in [2, 1, 4].

Note, all example scripts can be downloaded at the website:
`https://github.com/prmiles/mcmc_banana_examples`.

*Email: prmiles@ncsu.edu

2 Banana Example

For a full description of this example, see [3]. This example allows for an interesting comparison between different adaptive methods by constructing a non-Gaussian target distribution. This is accomplished by twisting the first two dimensions of the Gaussian distribution. As the Jacobian is 1, it makes it easy to calculate the right probability regions for the banana. The basics of the model are as follows.

$$B_f(x_i, a, b) = [ax_1, \frac{x_2}{a} - b(a^2x_1^2 + a^2)] \tag{1}$$

$$B_f^{-1}(x_i, a, b) = [x_1/a, ax_2 + ab(x_1^2 + a^2)] \tag{2}$$

We can construct a sum-of-squares function

$$B_{sos}(x_i, a, b, \mu_i, \lambda) = B_f^{-1}(x_i - \mu_i, a, b) \cdot \lambda \cdot B_f^{-1}(x_i - \mu_i, a, b) \tag{3}$$

where μ_i are the centers for our target distributions, λ is the target precision, and the banana parameters are a and b. For this particular example we consider the problem where we have 2 unknown parameters with center $\mu_i = 0$, and we have a target correlation of $\rho = 0.9$ for the first two dimensions. We subsequently define the target precision matrix λ

$$\lambda = \sigma^{-1} = \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}^{-1} \tag{4}$$

Calibration of this model yields a banana shaped pairwise correlation between x_1 and x_2 (hence the name).

Algorithm	Language		
	`Matlab`	`Python`	`R`
MH	208	207	205
AM	326	325	332
DR	1061	1075	1058
DRAM	1273	1261	1279

Table 1: In each language, the simulation was run 100 times. The numbers in the table represent the average number of accepted samples for each algorithm in each language.

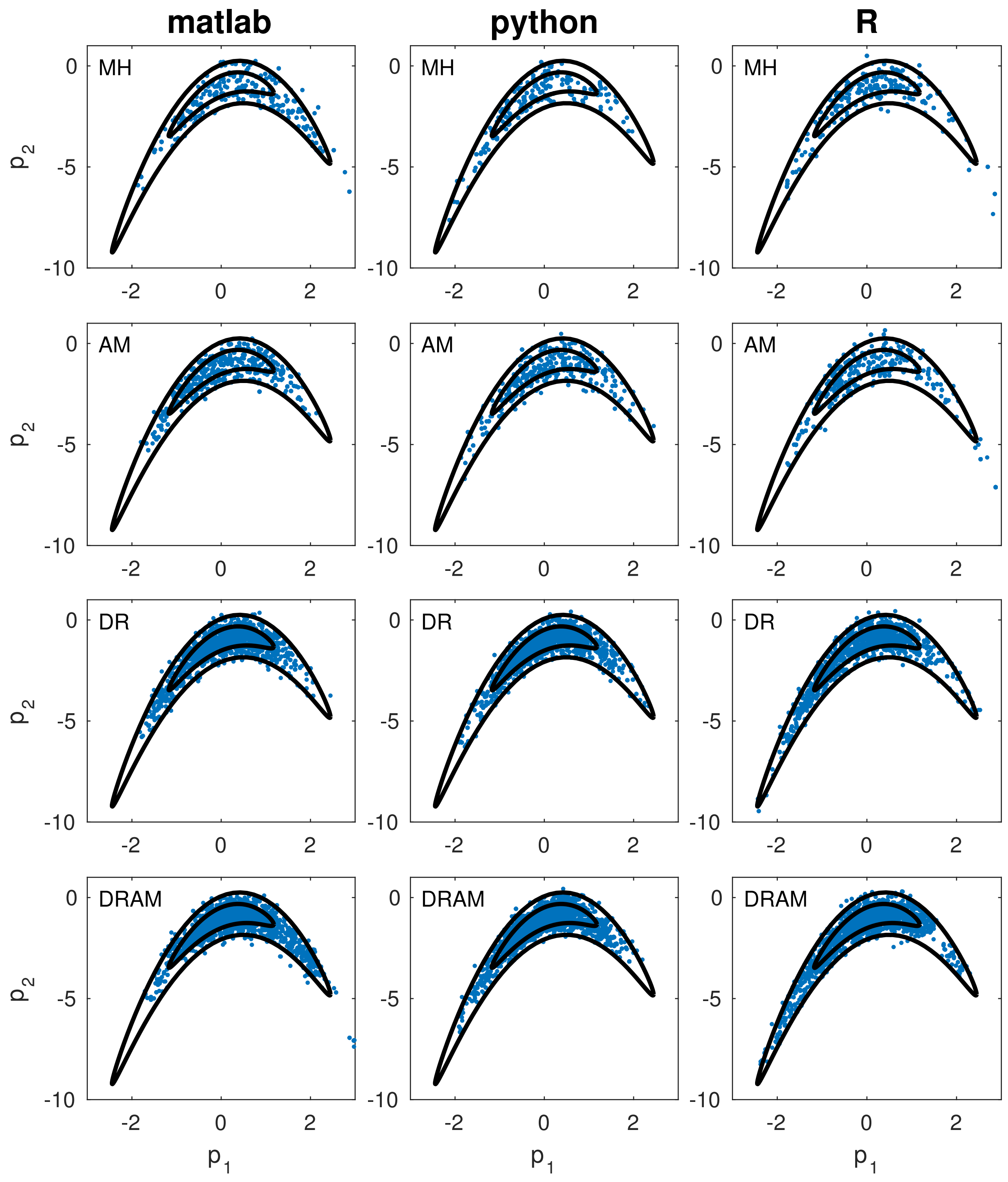

Figure 1: Comparison of pairwise correlation plot using different MCMC algorithms in different scripting languages.

3 Matlab

3.1 Key Features

To run MCMC simulations, the user must download the Toolbox from Marko Laine's website: http://helios.fmi.fi/~lainema/mcmc/. In order to access the toolbox, the user must add the directory to Matlab's search path.

```
% addpath to mcmcstat package
addpath('mcmcstat/');
```

Note that this command assumes you have placed the toolbox folder in the same directory as your script file. The required model functions are efficiently defined in Matlab

```
% 'banana' sum—of—squares
bananafun = @(x,a,b) [a.*x(:,1),x(:,2)./a-b.*((a.*x(:,1)).^2+a^2),x(:,3:end)];
bananainv = @(x,a,b) [x(:,1)./a,x(:,2).*a+a.*b.*(x(:,1).^2+a^2),x(:,3:end)];
bananass  = @(x,d) bananainv(x-d.mu,d.a,d.b)*d.lam*bananainv(x-d.mu,d.a,d.b)';
```

In order to run the MCMC simulations, we must first define several structures. The general procedure requires creation of a data structure, model parameter array, and settings specific to the MCMC simulation.

```
% Define data structure and model parameters array
% the data structure and parameters
data = struct('mu',mu,'a',a,'b',b,'lam',lam);
for i=1:npar
  params{i} = {sprintf('p_%d',i),0};
end

% Define model structure
model.ssfun     = bananass;
model.N         = 1;

% Define options structure with elements common to each sampling algorithm
options.nsimu    = 2000;
options.qcov     = eye(npar)*5; % [initial] proposal covariance
```

We can run different Metropolis algorithms by simply changing a few settings within the options structure

```
% Run Simulations
% Metropolis Hasting (MH)
options.method   = 'mh';
[mh.results,mh.chain] = mcmcrun(model,data,params,options);
% Adaptive Metropolis (AM)
options.method   = 'am';
options.adaptint = 100; % adaptation interval
[am.results,am.chain] = mcmcrun(model,data,params,options);
% Delayed Rejection (DR)
options.method   = 'dr';
options.ntry = 2; % number of DR steps
[dr.results,dr.chain] = mcmcrun(model,data,params,options);
% Delayed Rejection Adaptive Metropolis (DRAM)
options.method   = 'dram';
options.adaptint = 100; % adaptation interval
options.ntry = 2; % number of DR steps
[dram.results,dram.chain] = mcmcrun(model,data,params,options);
```

You can observe the sampling chains and correlation plots for any result by using the plotting routine within the mcmcstat toolbox.

```
% Chain panel
figure(1); clf
mcmcplot(dram.chain,[],dram.results.names,'chainpanel')

% Pairwise Correlation
```

```
figure(2); clf
mcmcplot(dram.chain,[1,2],dram.results.names,'pairs',0)
```

3.2 Complete Example Code

Listing 1: `Matlab` implementation for Banana example.

```
% setup workspace
clear; close all; clc;

% addpath to mcmcstat package
addpath('mcmcstat/');

% 'banana' sum-of-squares
bananafun = @(x,a,b) [a.*x(:,1),x(:,2)./a-b.*((a.*x(:,1)).^2+a^2),x(:,3:end)];
bananainv = @(x,a,b) [x(:,1)./a,x(:,2).*a+a.*b.*(x(:,1).^2+a^2),x(:,3:end)];
bananass  = @(x,d) bananainv(x-d.mu,d.a,d.b)*d.lam*bananainv(x-d.mu,d.a,d.b)';

a = 1; b = 1;          % banana parameters

npar = 2;              % number of unknowns
rho  = 0.9;             % target correlation
sig  = eye(npar); sig(1,2) = rho; sig(2,1) = rho;
lam  = inv(sig);        % target precision
mu   = zeros(1,npar);  % center

% Define data structure and model parameters array
% the data structure and parameters
data = struct('mu',mu,'a',a,'b',b,'lam',lam);
for i=1:npar
  params{i} = {sprintf('p_%d',i),0};
end

% Define model structure
model.ssfun     = bananass;
model.N         = 1;


% Define options structure with elements common to each sampling algorithm
options.nsimu    = 2000;
options.qcov     = eye(npar)*5; % [initial] proposal covariance

% Run Simulations
% Metropolis Hasting (MH)
options.method   = 'mh';
[mh.results,mh.chain] = mcmcrun(model,data,params,options);
% Adaptive Metropolis (AM)
options.method   = 'am';
options.adaptint = 100; % adaptation interval
[am.results,am.chain] = mcmcrun(model,data,params,options);
% Delayed Rejection (DR)
options.method   = 'dr';
options.ntry = 2; % number of DR steps
[dr.results,dr.chain] = mcmcrun(model,data,params,options);
% Delayed Rejection Adaptive Metropolis (DRAM)
options.method   = 'dram';
options.adaptint = 100; % adaptation interval
options.ntry = 2; % number of DR steps
[dram.results,dram.chain] = mcmcrun(model,data,params,options);

% Chain panel
figure(1); clf
```

```
mcmcplot(dram.chain,[],dram.results.names,'chainpanel')

% Pairwise Correlation
figure(2); clf
mcmcplot(dram.chain,[1,2],dram.results.names,'pairs',0)

% Print acceptance statistics
fprintf('\n----------------\n')
fprintf('MH: Number of accepted runs: %i out of %i (%4.2f%s)\n',length(unique(mh.chain
    (:,1))), options.nsimu, 100*(1-mh.results.rejected),'%');
fprintf('AM: Number of accepted runs: %i out of %i (%4.2f%s)\n',length(unique(am.chain
    (:,1))), options.nsimu, 100*(1-am.results.rejected),'%');
fprintf('DR: Number of accepted runs: %i out of %i (%4.2f%s)\n',length(unique(dr.chain
    (:,1))), options.nsimu, 100*(1-dr.results.rejected),'%');
fprintf('DRAM: Number of accepted runs: %i out of %i (%4.2f%s)\n',length(unique(dram.
    chain(:,1))), ...
     options.nsimu, 100*(1-dram.results.rejected),'%');

% save results for comparison
save('matlab_banana','mh','am','dr','dram');
```

4 Python

4.1 Key Features

To run MCMC simulations in `Python` one must install or download the package `pymcmcstat`. A complete tutorial for it's usage, including installation, can be found at https://prmiles.wordpress.ncsu.edu/. In our example script, we first import the required packages,

```python
# import required packages
from __future__ import division
import numpy as np
from pymcmcstat.MCMC import MCMC
from pymcmcstat.MCMCPlotting import MCMCPlotting
```

We define a series of functions as well as a class structure for communicating information within the MCMC simulation.

```python
# Define model object
def bananafunction(x, a, b):
    response = x
    response[:,0] = a*x[:,0]
    response[:,1] = x[:,1]*a**(-1) - b*((a*x[:,0])**(2) + a**2)
    return response

def bananainverse(x, a, b):
    response = x
    response[0] = x[0]*a**(-1)
    response[1] = x[1]*a + a*b*(x[0]**2 + a**2)
    return response

def bananass(theta, data):
    udobj = data.user_defined_object[0]
    lam = udobj.lam
    mu = udobj.mu
    a = udobj.a
    b = udobj.b
    npar = udobj.npar

    x = np.array([theta])
    x = x.reshape(npar,1)

    baninv = bananainverse(x-mu, a, b)

    stage1 = np.matmul(baninv.transpose(),lam)
    stage2 = np.matmul(stage1, baninv)

    return stage2

class Banana_Parameters:
    def __init__(self, rho = 0.9, npar = 12, a = 1, b = 1, mu = None):
        self.rho = rho
        self.npar = npar
        self.a = a
        self.b = b

        self.sig = np.eye(npar)
        self.sig[0,1] = rho
        self.sig[1,0] = rho
        self.lam = np.linalg.inv(self.sig)

        if mu is None:
            self.mu = np.zeros([npar, 1])
```

```
npar = 2 # number of model parameters
udobj = Banana_Parameters(npar = npar) # user defined object
```

We generate a different MCMC object for each algorithm we want to test and add the data to each object separately

```
# Initialize MCMC objects
mh = MCMC()
am = MCMC()
dr = MCMC()
dram = MCMC()

# initialize data within each MCMC object
mh.data.add_data_set(np.zeros(1),np.zeros(1), user_defined_object = udobj)
am.data.add_data_set(np.zeros(1),np.zeros(1), user_defined_object = udobj)
dr.data.add_data_set(np.zeros(1),np.zeros(1), user_defined_object = udobj)
dram.data.add_data_set(np.zeros(1),np.zeros(1), user_defined_object = udobj)
```

Similarly, we add model parameters to the simulation

```
# Add model parameters
for ii in xrange(npar):
    mh.parameters.add_model_parameter(name = str('$x_{}$'.format(ii+1)), theta0 = 0.0)
    am.parameters.add_model_parameter(name = str('$x_{}$'.format(ii+1)), theta0 = 0.0)
    dr.parameters.add_model_parameter(name = str('$x_{}$'.format(ii+1)), theta0 = 0.0)
    dram.parameters.add_model_parameter(name = str('$x_{}$'.format(ii+1)), theta0 = 0.0)
```

In order to define the specific algorithms to be used within the MCMC simulation, we must define the simulation options and also settings for the model

```
# Define options - include sampling algorithm!
mh.simulation_options.define_simulation_options(nsimu = int(2.0e3), qcov = np.eye(npar)
    *5, method='mh')
am.simulation_options.define_simulation_options(nsimu = int(2.0e3), qcov = np.eye(npar)
    *5, method='am', adaptint=100)
dr.simulation_options.define_simulation_options(nsimu = int(2.0e3), qcov = np.eye(npar)
    *5, method='dr', ntry = 2)
dram.simulation_options.define_simulation_options(nsimu = int(2.0e3), qcov = np.eye(npar)
    *5, method='dram', adaptint=100, ntry = 2)

# Define model settings
mh.model_settings.define_model_settings(sos_function = bananass, N = 1)
am.model_settings.define_model_settings(sos_function = bananass, N = 1)
dr.model_settings.define_model_settings(sos_function = bananass, N = 1)
dram.model_settings.define_model_settings(sos_function = bananass, N = 1)
```

We can now run the simulations

```
# Run simulation
mh.run_simulation()
am.run_simulation()
dr.run_simulation()
dram.run_simulation()
```

One can extract the results from each object, and as an example we have shown how to generate the chain panel and pairwise correlation plot for the results from the DRAM algorithm

```
# Extract results
mh_results = mh.simulation_results.results
am_results = am.simulation_results.results
dr_results = dr.simulation_results.results
dram_results = dram.simulation_results.results

# MCMC Plotting Routines
mcpl = MCMCPlotting()
```

```
# plot chain panel
mcpl.plot_chain_panel(dram_results['chain'],dram_results['names'])

# plot pairwise correlation
mcpl.plot_pairwise_correlation_panel(dram_results['chain'][:,0:2], dram_results['names'
    ][0:2])
```

4.2 Complete Example Code

Listing 2: `Python` implementation for Banana example.

```
# import required packages
from __future__ import division
import numpy as np
from pymcmcstat.MCMC import MCMC
from pymcmcstat.MCMCPlotting import MCMCPlotting

# Define model object
def bananafunction(x, a, b):
    response = x
    response[:,0] = a*x[:,0]
    response[:,1] = x[:,1]*a**(-1) - b*((a*x[:,0])**(2) + a**2)
    return response

def bananainverse(x, a, b):
    response = x
    response[0] = x[0]*a**(-1)
    response[1] = x[1]*a + a*b*(x[0]**2 + a**2)
    return response

def bananass(theta, data):
    udobj = data.user_defined_object[0]
    lam = udobj.lam
    mu = udobj.mu
    a = udobj.a
    b = udobj.b
    npar = udobj.npar

    x = np.array([theta])
    x = x.reshape(npar,1)

    baninv = bananainverse(x-mu, a, b)

    stage1 = np.matmul(baninv.transpose(),lam)
    stage2 = np.matmul(stage1, baninv)

    return stage2

class Banana_Parameters:
    def __init__(self, rho = 0.9, npar = 12, a = 1, b = 1, mu = None):
        self.rho = rho
        self.npar = npar
        self.a = a
        self.b = b

        self.sig = np.eye(npar)
        self.sig[0,1] = rho
        self.sig[1,0] = rho
        self.lam = np.linalg.inv(self.sig)
        if mu is None:
            self.mu = np.zeros([npar, 1])

npar = 2 # number of model parameters
udobj = Banana_Parameters(npar = npar) # user defined object

# Initialize MCMC objects
mh = MCMC()
am = MCMC()
dr = MCMC()
dram = MCMC()

# initialize data within each MCMC object
mh.data.add_data_set(np.zeros(1),np.zeros(1), user_defined_object = udobj)
am.data.add_data_set(np.zeros(1),np.zeros(1), user_defined_object = udobj)
dr.data.add_data_set(np.zeros(1),np.zeros(1), user_defined_object = udobj)
dram.data.add_data_set(np.zeros(1),np.zeros(1), user_defined_object = udobj)

# Add model parameters
for ii in xrange(npar):
    mh.parameters.add_model_parameter(name = str('$x_{}$'.format(ii+1)), theta0 = 0.0)
    am.parameters.add_model_parameter(name = str('$x_{}$'.format(ii+1)), theta0 = 0.0)
    dr.parameters.add_model_parameter(name = str('$x_{}$'.format(ii+1)), theta0 = 0.0)
    dram.parameters.add_model_parameter(name = str('$x_{}$'.format(ii+1)), theta0 = 0.0)

# Define options - include sampling algorithm!
mh.simulation_options.define_simulation_options(nsimu = int(2.0e3), qcov = np.eye(npar)
    *5, method='mh')
am.simulation_options.define_simulation_options(nsimu = int(2.0e3), qcov = np.eye(npar)
    *5, method='am', adaptint=100)
dr.simulation_options.define_simulation_options(nsimu = int(2.0e3), qcov = np.eye(npar)
    *5, method='dr', ntry = 2)
dram.simulation_options.define_simulation_options(nsimu = int(2.0e3), qcov = np.eye(npar)
    *5, method='dram', adaptint=100, ntry = 2)

# Define model settings
mh.model_settings.define_model_settings(sos_function = bananass, N = 1)
am.model_settings.define_model_settings(sos_function = bananass, N = 1)
dr.model_settings.define_model_settings(sos_function = bananass, N = 1)
dram.model_settings.define_model_settings(sos_function = bananass, N = 1)

# Run simulation
mh.run_simulation()
am.run_simulation()
dr.run_simulation()
dram.run_simulation()

# Extract results
mh_results = mh.simulation_results.results
am_results = am.simulation_results.results
dr_results = dr.simulation_results.results
dram_results = dram.simulation_results.results

# MCMC Plotting Routines
mcpl = MCMCPlotting()

# plot chain panel
mcpl.plot_chain_panel(dram_results['chain'],dram_results['names'])

# plot pairwise correlation
mcpl.plot_pairwise_correlation_panel(dram_results['chain'][:,0:2], dram_results['names'
    ][0:2])
```

5 R

5.1 Key Features

To run MCMC simulations in `R` one must install the `FME` package. This can be done from CRAN using

```
install.packages('FME')
```

We define a series of functions to generate our sum-of-squares value, which is a required input to the MCMC simulation.

```
# define inverse banana function
inv_banana <- function(y1,y2,a=1,b=1) {
  x1 = y1/a
  x2 = a*(y2 + b*((a*x1^2)+a^2))
  return(c(x1,x2))
}

# define function to be called in MCMC simulation
fme_banana <- function(y1,y2) {
  return(inv_banana(y1,y2)[2])
}

# Define a function that estimates the probability of a multinormally distributed vector
pmultinorm <- function(vec, mean, Cov) {
  diff <- vec - mean
  ex   <- -0.5*t(diff) %*% solve(Cov) %*% diff
  rdet   <- sqrt(det(Cov))
  power  <- -length(diff)*0.5
  return((2.*pi)^power / rdet * exp(ex))
}


# Define the target function which returns the -2*log(probability) of the value
BananaSS <- function (p)
{
  P <- c(p[1], fme_banana(p[1], p[2]))
  Cov <- matrix(nr = 2, data = c(1, 0.9, 0.9, 1))
  -2*sum(log(pmultinorm(P, mean = 0, Cov = Cov)))
}
```

We can then run different types of MCMC simulations by changing whether or not we include key words such as `updatecov` and `ntrydr`.

```
# Run MCMC Simulations
# Metropolis Hastings
MCMC_MH <- modMCMC(f = BananaSS, p = c(0, 0.5), jump = diag(nrow = 2, x = 5), niter =
    2000)
MCMC_MH$count

# Adaptive Metropolis
MCMC_AM <- modMCMC(f = BananaSS, p = c(0, 0.5), jump = diag(nrow = 2, x = 5), updatecov =
     100, niter = 2000)
MCMC_AM$count

# Delayed Rejection
MCMC_DR <- modMCMC(f = BananaSS, p = c(0, 0.5), jump = diag(nrow = 2, x = 5), ntrydr = 2,
     niter = 2000)
MCMC_DR$count

# Delayed Rejection Adaptive Metropolis
print(system.time(
  MCMC_DRAM <- modMCMC(f = BananaSS, p = c(0, 0.5), jump = diag(nrow = 2, x = 5),
      updatecov = 100, ntrydr = 2, niter = 2000)
))
```

The results from the simulation can be observed in several different ways. The raw chains can be plotted and also pairwise correlations via

```
# Plot Chains
par(mfrow = c(4, 2))
par(mar = c(2, 2, 4, 2))
plot(MCMC_MH, mfrow = NULL, main = "MH")
plot(MCMC_AM, mfrow = NULL, main = "AM")
plot(MCMC_DR, mfrow = NULL, main = "DR")
plot(MCMC_DRAM, mfrow = NULL, main = "DRAM")
mtext(outer = TRUE, side = 3, line = -2, at = c(0.05, 0.95),
      c("y1", "y2"), cex = 1.25)
par(mar = c(5.1, 4.1, 4.1, 2.1))


# Plot Pairwise Correlation
par(mfrow = c(2, 2))
xl <- c(-3, 3)
yl <- c(-8, 1)
plot(MCMC_MH$pars,  main = "MH", xlim = xl, ylim = yl)
plot(MCMC_AM$pars, main = "AM", xlim = xl, ylim = yl)
plot(MCMC_DR$pars, main = "DR", xlim = xl, ylim = yl)
plot(MCMC_DRAM$pars, main = "DRAM", xlim = xl, ylim = yl)
```

5.2 Complete Example Code

Listing 3: R implementation for Banana example.

```
# add/install required package
# install.package('FME')
library(FME) # package for running MCMC simulations

# define inverse banana function
inv_banana <- function(y1,y2,a=1,b=1) {
  x1 = y1/a
  x2 = a*(y2 + b*((a*x1^2)+a^2))
  return(c(x1,x2))
}

# define function to be called in MCMC simulation
fme_banana <- function(y1,y2) {
  return(inv_banana(y1,y2)[2])
}

# Define a function that estimates the probability of a multinormally distributed vector
pmultinorm <- function(vec, mean, Cov) {
  diff <- vec - mean
  ex   <- -0.5*t(diff) %*% solve(Cov) %*% diff
  rdet   <- sqrt(det(Cov))
  power  <- -length(diff)*0.5
  return((2.*pi)^power / rdet * exp(ex))
}


# Define the target function which returns the -2*log(probability) of the value
BananaSS <- function (p)
{
  P <- c(p[1], fme_banana(p[1], p[2]))
  Cov <- matrix(nr = 2, data = c(1, 0.9, 0.9, 1))
```

```r
  -2*sum(log(pmultinorm(P, mean = 0, Cov = Cov)))
}


# Run MCMC Simulations
# Metropolis Hastings
MCMC_MH <- modMCMC(f = BananaSS, p = c(0, 0.5), jump = diag(nrow = 2, x = 5), niter =
    2000)
MCMC_MH$count

# Adaptive Metropolis
MCMC_AM <- modMCMC(f = BananaSS, p = c(0, 0.5), jump = diag(nrow = 2, x = 5), updatecov =
     100, niter = 2000)
MCMC_AM$count

# Delayed Rejection
MCMC_DR <- modMCMC(f = BananaSS, p = c(0, 0.5), jump = diag(nrow = 2, x = 5), ntrydr = 2,
     niter = 2000)
MCMC_DR$count

# Delayed Rejection Adaptive Metropolis
print(system.time(
  MCMC_DRAM <- modMCMC(f = BananaSS, p = c(0, 0.5), jump = diag(nrow = 2, x = 5),
      updatecov = 100, ntrydr = 2, niter = 2000)
))
MCMC_DRAM$count

# Plot Chains
par(mfrow = c(4, 2))
par(mar = c(2, 2, 4, 2))
plot(MCMC_MH, mfrow = NULL, main = "MH")
plot(MCMC_AM, mfrow = NULL, main = "AM")
plot(MCMC_DR, mfrow = NULL, main = "DR")
plot(MCMC_DRAM, mfrow = NULL, main = "DRAM")
mtext(outer = TRUE, side = 3, line = -2, at = c(0.05, 0.95),
      c("y1", "y2"), cex = 1.25)
par(mar = c(5.1, 4.1, 4.1, 2.1))


# Plot Pairwise Correlation
par(mfrow = c(2, 2))
xl <- c(-3, 3)
yl <- c(-8, 1)
plot(MCMC_MH$pars,  main = "MH", xlim = xl, ylim = yl)
plot(MCMC_AM$pars, main = "AM", xlim = xl, ylim = yl)
plot(MCMC_DR$pars, main = "DR", xlim = xl, ylim = yl)
plot(MCMC_DRAM$pars, main = "DRAM", xlim = xl, ylim = yl)

# Write chains to text file
write.table(MCMC_MH$pars, file="mh.txt", sep = ',', row.names=FALSE, col.names=FALSE)
write.table(MCMC_AM$pars, file="am.txt", sep = ',', row.names=FALSE, col.names=FALSE)
write.table(MCMC_DR$pars, file="dr.txt", sep = ',', row.names=FALSE, col.names=FALSE)
write.table(MCMC_DRAM$pars, file="dram.txt", sep = ',', row.names=FALSE, col.names=FALSE)
```

References

[1] Heikki Haario, Marko Laine, Antonietta Mira, and Eero Saksman. Dram: efficient adaptive mcmc. *Statistics and Computing*, 16(4):339–354, 2006.

[2] Heikki Haario, Eero Saksman, Johanna Tamminen, et al. An adaptive metropolis algorithm. *Bernoulli*, 7(2):223–242, 2001.

[3] Marko Laine et al. *Adaptive MCMC methods with applications in environmental and geophysical models*. Finnish Meteorological Institute, 2008.

[4] Ralph C Smith. *Uncertainty quantification: theory, implementation, and applications*, volume 12. Siam, 2013.