Maximum Likelihood Estimation of Linear Models using general purpose optimization algorithms

Dominic Noy

April 2017

Similar alogrithms are implemented in R and Python.

Univariate Normal Distribution:

$$
\begin{aligned}
&X \sim NV(\mu, \sigma^2) \\
&f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}
\end{aligned} \tag{1}
$$

Likelihood Function:

$$
L(\mu, \sigma^2; x_1, \ldots, x_n) = (2\pi\sigma^2)^{-}\frac{n}{2} e^{-\frac{1}{2\sigma^2}\sum(x-\mu)^2} \tag{2}
$$

Log Likelihood Function:

$$
l(\mu, \sigma^2; x_1, \ldots, x_n) = \sum(-\frac{1}{2} log\, 2\pi - \frac{1}{2} log\, \sigma^2 - \frac{(X_i-\mu)^2}{2\sigma^2}) = -\frac{n}{2} ln(2\pi) - \frac{n}{2} ln(\sigma^2) - \frac{1}{2\sigma^2}\sum(x-\mu)^2 \tag{3}
$$

R code

```
#Simulate Data
set.seed(1235)
mu=2; sigma=1
x = rnorm(100,mu,sigma)

#Likelihood Function
MLEUni <- function(x, par)
  {
  n<-length(x)
  -(-(n/2)*log(2*pi)-(n/2)*log(par[2]^2)-(1/(2*par[2]^2))*sum((x-par[1])^2))
  }

par <- c(0,1)

#Otimizer, Method = Nelder-Mead
estimate = optim(par, MLEUni, x=x, hessian=TRUE, method="Nelder-Mead")

#Print Results
estimate = data.frame(cbind(c(estimate$par[1], estimate$par[2]), c(mu, sigma)));
rownames(estimate) = c("mu", "sigma")
colnames(estimate) = c("estimate", "true")
estimate
```

```
##       estimate true
## mu    2.077866    2
## sigma 1.045054    1
```

Python Code

```
# import the packages
import numpy as np
from scipy.optimize import minimize
import scipy.stats as stats
import time


#Simulate data
mu = 5
sigma = 15
N = 100
x = np.random.normal(mu, sigma, N)

#Likelihood Function
def MLEUni(params):
    n=len(x)
    logLik = -(-(n/2)*np.log(2*np.pi)-(n/2)*np.log(params[1]**2)
    -(1/(2*params[1]**2))*sum((np.array(x)-params[0])**2))
    return(logLik)

#Optimizer
initParams = [0, 1]
results = minimize(MLEUni, initParams, method='nelder-mead')

#Print Results
estimated_parameters = {"mu":round(results.x[0],3), "sigma":round(results.x[1],3)}
parameters = {"mu":mu, "sigma":sigma}
print('estimated')
print(estimated_parameters)
print('true')
print(parameters)
```

```
## estimated
## {'mu': 4.924, 'sigma': 13.939}
## true
## {'mu': 5, 'sigma': 15}
```

Bivariate Normal Distribution:

$$\begin{aligned}
&X_1, X_2 \sim MNV(\mu, \Sigma) \\
&\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \\
&\Sigma = \begin{pmatrix} \sigma_1 & \rho \\ \rho & \sigma_2 \end{pmatrix} \\
&f(x_1, x_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} e^{-\frac{z}{2(1-\rho^2)}} \\
&z = \frac{(x_1-\mu_1)^2}{\sigma_1^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2}
\end{aligned} \tag{4}$$

Likelihood Function:

$$\begin{aligned}
l(\mu, \Sigma; x_1, ..., x_n) = &-n(log(1) + log(2\pi) + log(\sigma_1) + log(\sigma_2) + 0.5log(1-\rho^2)) \\
&-\frac{0.5}{(1-\rho^2)}(\frac{\sum(x_1-\mu_1)^2}{\sigma_1^2} + (\frac{\sum(x_2-\mu_2)^2}{\sigma_2^2} - 2\rho\frac{\sum(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1^2\sigma_2^2})
\end{aligned} \tag{5}$$

R code

```r
#Generate Data
N <- 1000
mu=c(10,10); sig1=1; sig2=2; rho=0.5; sigma<-matrix(c(sig1,rho,rho,sig2), byrow=T, ncol=2)
y <- MASS::mvrnorm(n=N, mu=mu, Sigma=sigma)


#log likelihood of bivariate nv
#-n*(log(1)+log(2*pi)+log(sig1) + log(sig2) + 0.5*log(1-rho^2)) -
#0.5/(1-rho^2)*(sum((x1-mu1)^2)/sig1^2 + sum((x2-mu2)^2)/sig2^2 -
#2*rho*sum((x1-mu1)*(x2-mu2))/(sig1*sig2) )

#Likelihood Function
MLEBi <- function(parms)
  {
  mu1  = parms[1]; mu2  = parms[2]; sig1 = parms[3]; sig2 = parms[4]; rho  = parms[5];
  x1 <- y[,1]; x2 <- y[,2]; n = length(x1)

  loglik = as.numeric(sum(mvtnorm::dmvnorm(y, mean = c(mu1,mu2), sigma =
           matrix(c(sig1,rho,rho,sig2), nrow=2, byrow=T), log = T)))

  return(-loglik)
  -loglik
  }

#optim algorithm
eps <- 2.220446e-16

#Note that computational problems may arise due to the possible multimodality of the
#likelihood function("the optimization method L-BFGS-B of the function optim" could not be
#recommendable; and it would be better to use the R package DEoptim)".
estimate<-DEoptim::DEoptim(fn=MLEBi, lower=c(-100,-100, eps,eps, -1+eps),
```

```r
                           upper=c(100,100, 100,100, 1-eps),
                           DEoptim::DEoptim.control(trace=FALSE))
estimate<-data.frame(cbind(estimate$optim$bestmem, c(mu[1], mu[2], sig1, sig2, rho)))

colnames(estimate)<-c("estimate", "true")
rownames(estimate)<-c("mu1", "mu2", "sig1", "sig2", "rho")
estimate
```

```
##        estimate true
## mu1  10.0111105 10.0
## mu2  10.0243365 10.0
## sig1  0.9484815  1.0
## sig2  2.1219386  2.0
## rho   0.5660109  0.5
```

Linear Model

Univariate

R code

```r
#Generate Data
N<-100
x = runif(N)

beta0<-5
beta1<-20
mu=0
sigma=1
y <- beta1 * x + beta0 + rnorm(N, mu, sigma)


linMLEuni <- function(par)
  {
  beta0<-par[1]
  beta1<-par[2]
  mu<-par[3]
  sigma<-par[4]

  R = y - x * beta1 - beta0
  n<-length(R)
  -(-(n/2)*log(2*pi)-(n/2)*log(sigma^2)-(1/(2*sigma^2))*sum((R-mu)^2))
  }

#Optimizer
par <- c(0,2, 0,1)
estimate<-optim(par,linMLEuni, hessian=T, method="L-BFGS-B",
                lower=c(-Inf,-Inf, 0, 0.999999),
                upper=c(Inf,Inf,0.0001,1))

estimate<-data.frame(cbind(estimate$par, c(beta0, beta1, mu, sigma)))
colnames(estimate)<-c("estimate", "true")
```

```r
rownames(estimate)<-c("beta0", "beta1", "mu", "sigma")
estimate
```

```
##        estimate true
## beta0  5.041377    5
## beta1 19.784493   20
## mu     0.000000    0
## sigma  0.999999    1
```

Python Code

```python
# Import the packages
import numpy as np
from scipy.optimize import minimize
import scipy.stats as stats
import time


# Generate Data
N=1000
x = np.random.random((N,))
beta0 = 20
beta1 = 30
y = beta1*x + beta0 + np.random.standard_normal((N,))


def MLEBi(params):
    beta0 = params[0]
    beta1 = params[1]
    mu = params[2]
    sigma = params[3]

    R = y - x * beta1 - beta0

    n=len(R)
    loglik=-(n/2)*np.log(2*np.pi)-(n/2)*np.log(sigma**2)-(1/(2*sigma**2))*sum((R-mu)**2)
    return(-loglik)




#Optimizer
initParams = [0.5, 0.5, 0, 1]
eps=2.220446e-16
bnds = ((-float("inf"),float("inf")), (-float("inf"), float("inf")), (0, eps), (1-eps, 1))
results = minimize(MLEBi, initParams, method='SLSQP', bounds=bnds, tol=1e-10)
#TNC and L-BFGS-B both support bound constraints (e.g. x[0] >= 0)
#SLSQP is more flexible, supporting any combination of bounds,
#equality and inequality-based constraints.


# Print the results. They should be really close to your actual values
print results.x
```

```
beta0_est=round(results.x[0],3)
beta1_est=round(results.x[1],3)
mu_est=round(results.x[2],3)
sigma_est=round(results.x[3],3)

estimated_parameters = {"beta0":beta0_est, "beta1":beta1_est, "mu":mu_est,
"sigma":sigma_est}
parameters = {"beta0":beta0, "beta1":beta1,"mu":0, "sigma":1}
print('estimated')
print(estimated_parameters)
print('true')
print(parameters)
```

```
## [  7.29369294e+01   5.12664525e+01  -2.50624176e-02  -1.10204960e+00]
## estimated
## {'mu': -0.025, 'sigma': -1.102, 'beta1': 51.266, 'beta0': 72.937}
## true
## {'mu': 0, 'sigma': 1, 'beta1': 30, 'beta0': 20}
```

Bivariate Linear Model - Matrix Notation

$$
\begin{aligned}
&y_i = \beta_0 + \beta_1 x_{i1} + \mu_i, i = 1, 2\\
&Y = XB + \mu\\
&\begin{vmatrix} y1 \\ \vdots \\ yn \end{vmatrix} = \begin{vmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & xn \end{vmatrix} \begin{vmatrix} \beta 0 \\ \beta 1 \end{vmatrix} + \begin{vmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{vmatrix}\\
&\epsilon \sim MVN(\mu, \Sigma)\\
&\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \Sigma = \begin{bmatrix} \sigma_1 & \rho \\ \rho & \sigma_2 \end{bmatrix}
\end{aligned}
\tag{6}
$$

R code

```
#Generate Data
N<-1000; beta0=-10; beta1=10; sig1<-1; sig2<-4; rho<-0.2; mu1<-0; mu2<-0
Sigma<-matrix(c(sig1,rho,rho,sig2), nrow=2, byrow=T) #this is the variance
x = runif(N)
y <- beta1 * x + beta0 + MASS::mvrnorm(N, c(mu1,mu2), Sigma)

#Likelihood Function
linMLE_Bi <- function(par)
  {
  beta0<-par[1]; beta1<-par[2]; sig1<-par[3]; sig2<-par[4]; rho<-par[5]
   n=length(x)

  # Find residuals
  r = y - x * beta1 - beta0

  loglik2 = -as.numeric(sum(mvtnorm::dmvnorm(r, mean = c(0,0), sigma =
            matrix(c(sig1,rho,rho,sig2), nrow=2, byrow=T),log = T)))
```

```r
  }

#Optimization
estimate<-DEoptim::DEoptim(fn=linMLE_Bi, lower=c(-50,-50,  0.00001,0.00001, 0.00001),
                           upper=c(50,50, 100,100, 0.99999),
                           DEoptim::DEoptim.control(trace=FALSE))

#Print Results
estimate_par<-estimate$optim$bestmem
names(estimate_par)<-c("beta0","beta1", "sigma1", "sigma2", "rho")
estimate<-data.frame(cbind(estimate_par, c(beta0, beta1, sig1, sig2, rho)))
colnames(estimate)<-c("estimate", "true")
estimate
```

```
##          estimate  true
## beta0  -9.9882788 -10.0
## beta1   9.9728507  10.0
## sigma1  0.9428424   1.0
## sigma2  3.9793011   4.0
## rho     0.1817027   0.2
```

Python Code

```python
# Import the packages
import numpy as np
from scipy.optimize import minimize
import scipy.stats as stats
import time


# Generate Data
N=1000; beta0 = -20; beta1 = 30; var1 = 2; var2 = 3; cov = 0.5

Sigma = [[2,0.5],[0.5,3]]; mean = [0, 0]

x = np.random.random((N,))
y = beta1*x
y2= beta0+np.random.multivariate_normal(mean, Sigma, N)
y3=y2 + y[:,np.newaxis]

#Likelihood Function
def linMLE_Bi(params):
    beta0 = params[0]
    beta1 = params[1]
    sig1 = params[2]
    sig2 = params[3]
    rho = params[4]

    r1= x * beta1 + beta0
    r2 = y3 - r1[:,np.newaxis]

    n=len(x)
    aux1 = (np.log(sig1) + np.log(sig2) + 0.5 * np.log(1-rho**2))
```

```
    aux2 = sum((r2[:,0])**2) / sig1**2 + sum((r2[:,1])**2)
    aux3 = (aux2 / sig2**2 - 2*rho * sum((r2[:,0]) * (r2[:,1])) / (sig1*sig2))
    loglik = -n * aux1 - 0.5 / (1-rho**2) * aux3
    return(-loglik)


#Optimization
initParams = [0.5, 0.5, 0.5, 0.5, 0.5]
eps=2.220446e-16
bnds = ((-float("inf"),float("inf")), (-float("inf"), float("inf")), (eps, 100),
(eps, 100),(-1+eps, 1-eps))
results = minimize(linMLE_Bi, initParams, method='SLSQP', bounds=bnds, tol=1e-10)

#Print Results
estimated_parameters = {"beta0":round(results.x[0],3), "beta1":round(results.x[1],3),
"var1":round(results.x[2]**2,3), "var2":round(results.x[3]**2,3),
"cov":round(results.x[4]*results.x[2]*results.x[3], 3)}
parameters = {"beta0":beta0, "beta1":beta1, "var1":var1, "var2":var2, "cov":cov}
print 'estimated'
print estimated_parameters
print 'true'
print parameters
```

```
## -c:30: RuntimeWarning: invalid value encountered in log
## estimated
## {'var2': nan, 'var1': nan, 'cov': nan, 'beta1': nan, 'beta0': nan}
## true
## {'var2': 3, 'var1': 2, 'cov': 0.5, 'beta1': 30, 'beta0': -20}
```

Multivariate Linear Model (for n=3)

$$
\begin{gathered}
y_i = \beta_0 + \beta_1 x_{i1} + \mu_i, i = 1,2,3 \\
Y = XB + \mu \\
\begin{vmatrix} y_{11} \\ y_{12} \\ y_{13} \\ \vdots \\ y_{m1} \\ y_{m2} \\ y_{m3} \end{vmatrix} = \begin{vmatrix} 1 & x^1_{11} \\ 1 & x^1_{12} \\ 1 & x^1_{13} \\ \vdots & \vdots \\ 1 & x^1_{m1} \\ 1 & x^1_{m2} \\ 1 & x^1_{m3} \end{vmatrix} \begin{vmatrix} \beta 0 \\ \beta 1 \end{vmatrix} + \begin{vmatrix} \epsilon_{11} \\ \epsilon_{12} \\ \epsilon_{13} \\ \vdots \\ \epsilon_{m1} \\ \epsilon_{m2} \\ \epsilon_{m3} \end{vmatrix} \\
\epsilon_i \sim MVN(\mu_i, Vi) \\
f(x) = \frac{1}{\sqrt{(2\pi)^k \det \Sigma}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right) \\
\mu_i = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, Vi = \begin{bmatrix} \sigma_1 & \rho_{12} & \rho_{13} \\ . & \sigma_2 & \rho_{23} \\ . & . & \sigma_3 \end{bmatrix}
\end{gathered} \tag{7}
$$

Note: the optimization algorithms may take a few minutes. ###R code

```
#Generate Data
N<-1000; beta0=-10; beta1=10; var1 = 3; var2 = 2; var3 = 4; cov12 = 1.5; cov13 = 0.5;
```

```r
cov23 = 0.9
Sigma<-rbind(c(var1,cov12,cov13), c(cov12,var2,cov23), c(cov13,cov23,var3))
x = runif(N)
y <- beta1 * x + beta0 + MASS::mvrnorm(N, c(0,0,0), Sigma)

#Likelihood Function
linMLE_MVN <- function(par)
  {
  beta0<-par[1]; beta1<-par[2]; sig1<-par[3]; sig2<-par[4]; sig3<-par[5]; rho12<-par[6];
  rho13<-par[7]; rho23<-par[8]; n=length(x)
  # Find residuals
  r = y - x * beta1 - beta0
  loglik2=-as.numeric(sum(mvtnorm::dmvnorm(r, mean = c(0,0,0), sigma =
           cbind(c(sig1, rho12,rho13), c(rho12, sig2, rho23), c(rho13, rho23, sig3)),
           log = T)))
  }

#Optimization
estimate<-DEoptim::DEoptim(fn = linMLE_MVN, DEoptim::DEoptim.control(NP = 80,
          itermax = 200,F = 1.2, CR = 0.7, trace =FALSE), lower=c(-50,-50,
          0.00001,0.00001, 0.00001,0.00001,0.00001, 0.00001),
          upper=c(50,50, 100,100,100, 0.99999, 0.99999, 0.99999))

#Print Results
estimate_par<-estimate$optim$bestmem
names(estimate_par)<-c("beta0","beta1", "var1", "var2","var3", "cov12", "cov13", "cov23")
estimate<-data.frame(cbind(estimate_par, c(beta0, beta1, var1, var2,
                                           var3, cov12, cov13, cov23)))
names(estimate)<-c("estimate","true")
estimate
```

```
##         estimate  true
## beta0 -9.8708534 -10.0
## beta1  9.8792405  10.0
## var1   2.4646768   3.0
## var2   1.6528425   2.0
## var3   3.9140302   4.0
## cov12  0.9983861   1.5
## cov13  0.2042776   0.5
## cov23  0.7397904   0.9
```

Python code

```python
import numpy as np
import math
import scipy.sparse as sp
import scipy.sparse.linalg as spln
from scipy.optimize import minimize
import scipy.stats as stats
import time


# Generate Data
```

```python
N=1000
beta0 = -20; beta1 = 30; var1 = 2.3; var2 = 2.5; var3 = 4; cov12 = 0.5; cov13 = 0.9;
cov23 = 0.2
sigma = np.matrix([[var1, cov12, cov13],
           [cov12, var2, cov23],
           [cov13, cov23, var3]
          ])
mu = np.array([0,0,0])
x = np.random.random((N,))
y = beta1*x
y2= beta0+np.random.multivariate_normal(mu, sigma, N)
y3=y2 + y[:,np.newaxis]


#Likelihood Function
def linMLE_MVN(params):
    beta0 = params[0];beta1 = params[1];sig1 = params[2];sig2 = params[3];sig3 = params[4];
    rho12 = params[5];rho13 = params[6];rho23 = params[7]

    r1= x * beta1 + beta0
    r2 = y3 - r1[:,np.newaxis]
    sigma = np.matrix([[sig1, rho12, rho13],[rho12, sig2, rho23],[rho13, rho23, sig3]])

    def MLE(x):
      aux1 = np.exp(-0.5*np.dot(np.dot((x).T,sigma.I),(x)))
      aux2 = np.sqrt(np.linalg.det(2*np.pi*sigma))
      mvn = aux1 / aux2

      return(mvn)

    loglik=sum(np.log(np.apply_along_axis(MLE, 1, r2))) #for matrix
    return(-loglik)



# Optimization
initParams = [5, -10, 5, 3, 10, 0.3, 0.3, 0.3]
bnds = ((-50,50), (-50, 50), (2, 100), (2, 100), (2, 100), (-0.99, 0.99),
(-0.99, 0.99), (-0.99, 0.99))
results = minimize(linMLE_MVN, initParams, method='SLSQP', bounds=bnds, tol=0.1)


# Print Results
print 'estimated'
estimated_betas = {"beta0":round(results.x[0],3), "beta1":round(results.x[1],3)}
estimated_variance = {"var1":round(results.x[2],3), "var2":round(results.x[3],3),
"var3":round(results.x[4],3)}
estimated_covariance = {"cov12":round(results.x[5],3), "cov13":round(results.x[6],3),
"cov23":round(results.x[7],3)}
print estimated_betas
print estimated_variance
print estimated_covariance
print 'true'
```

```
beta = {"beta0":beta0, "beta1":beta1}
variance = {"var1":var1, "var2":var2, "var3":var3}
covariance = {"cov12":cov12, "cov13":cov13, "cov23":cov23}
print beta
print variance
print covariance
```

```
## estimated
## {'beta1': 29.878, 'beta0': -19.971}
## {'var1': 2.213, 'var3': 3.994, 'var2': 2.526}
## {'cov12': 0.446, 'cov13': 0.918, 'cov23': 0.262}
## true
## {'beta1': 30, 'beta0': -20}
## {'var1': 2.3, 'var3': 4, 'var2': 2.5}
## {'cov12': 0.5, 'cov13': 0.9, 'cov23': 0.2}
```

Multivariate Linear Model with specified covariance matrix for t = 3 (= extended Linear Model)

$\gamma_Z(0) = 2\sigma_M^2 + \sigma_T^2$,

$\gamma_Z(1) = -\sigma_M^2$,

$\gamma_Z(j) = 0$, for $j > 1$,

$\Sigma = \gamma_Z(0)I + \gamma_Z(1)\Delta$

$$\Delta = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 1 & \ddots & \vdots & 0 \\ 0 & 1 & 0 & \ddots & 0 & \vdots \\ \vdots & 0 & \ddots & 0 & 1 & 0 \\ 0 & \vdots & \ddots & 1 & 0 & 1 \\ 0 & 0 & \cdots & 0 & 1 & 0 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} \gamma_Z(0)\gamma_Z(1) & -\gamma_Z(1) & 0 & \cdots & 0 & 0 \\ -\gamma_Z(1) & \gamma_Z(0)\gamma_Z(1) & -\gamma_Z(1) & \ddots & \vdots & 0 \\ 0 & -\gamma_Z(1) & \gamma_Z(0)\gamma_Z(1) & \ddots & 0 & \vdots \\ \vdots & 0 & \ddots & \gamma_Z(0)\gamma_Z(1) & -\gamma_Z(1) & 0 \\ 0 & \vdots & \ddots & -\gamma_Z(1) & \gamma_Z(0)\gamma_Z(1) & -\gamma_Z(1) \\ 0 & 0 & \cdots & 0 & -\gamma_Z(1) & \gamma_Z(0)\gamma_Z(1) \end{bmatrix}$$

R code

```r
# Generate Data
N<-1000; beta0=30; beta1=-20
sigmaM<-5
sigmaT<-10
Sigma<-rbind(c(2*sigmaM^2+sigmaT^2,-sigmaM^2,0), c(-sigmaM^2,2*sigmaM^2+sigmaT^2,-sigmaM^2),
             c(0,-sigmaM^2,2*sigmaM^2+sigmaT^2))

x = runif(N)
y <- beta1 * x + beta0 + MASS::mvrnorm(N, c(0,0,0), Sigma)
```

```
# Likelihood Function
linMLE_MVN = function(par)
  {
  beta0 = par[1]; beta1<-par[2]; sigmaM<-par[3]; sigmaT<-par[4]; n=length(x)
  # Find residuals
  r = y - x * beta1 - beta0

  loglik2 = -as.numeric(sum(mvtnorm::dmvnorm(r, mean = c(0,0,0), sigma =
            rbind(c(2*sigmaM^2+sigmaT^2,-sigmaM^2,0),
            c(-sigmaM^2,2*sigmaM^2+sigmaT^2,-sigmaM^2),
            c(0,-sigmaM^2,2*sigmaM^2+sigmaT^2)) , log = T)))
  }



# Optimization
estimate = DEoptim::DEoptim(fn=linMLE_MVN, DEoptim::DEoptim.control(NP = 80, itermax = 200,
           F = 1.2, CR = 0.9, trace =FALSE), lower=c(0,-1000,  0.00001,0.00001),
           upper=c(1000,0.00001, 1000,1000))



# Print Results
estimate_par<-estimate$optim$bestmem
names(estimate_par)<-c("beta0","beta1", "sigmaM", "sigmaT")
estimate<-data.frame(cbind(estimate$optim$bestmem, c(beta0, beta1, sigmaM, sigmaT)))
colnames(estimate)<-c("estimate", "true")
rownames(estimate)<-c("beta0", "beta1", "sigmaM", "sigmaT")
estimate
```

```
##           estimate true
## beta0    29.851215   30
## beta1   -19.188685  -20
## sigmaM    4.717732    5
## sigmaT   10.237496   10
```

t = variable (10)

```
#Generate Data
N<-100; beta0=180; beta1=-20

n<-10 #size of the matrix (how many steps)
delta<-matrix(rep(0, n*n), nrow=n, ncol=n)

for(i in 1:(n )){
  for (j in 1:n)
    if (i == j+1 | i == j-1){
      delta[i,j] <- 1;
    }
}

#create symmetric covariance matrix (with rho)
sigmaM<-5
```

```r
sigmaT<-10
Sigma<-diag(x = 2*sigmaM^2+sigmaT^2, nrow=n, ncol=n)-sigmaM^2*delta

x = runif(N)
y <- beta1 * x + beta0 + MASS::mvrnorm(N, rep(0,n), Sigma)


#Likelihood Function
linMLE_MVN = function(par)
{
  beta0 = par[1]; beta1<-par[2]; sigmaM<-par[3]; sigmaT<-par[4]
  # Find residuals
  r = y - x * beta1 - beta0

  mu<-rep(0,n)
  sig<-diag(x = 2*sigmaM^2+sigmaT^2, nrow=n, ncol=n)+-sigmaM^2*delta
  loglik2 = -as.numeric(sum(mvtnorm::dmvnorm(r, mean = mu,sigma = sig, log = T)))
  loglik2
  }


#Optimization
estimate = DEoptim::DEoptim(fn = linMLE_MVN, DEoptim::DEoptim.control(NP = 80,
           itermax = 200, F = 1.2, CR = 0.7, trace =FALSE),
           lower=c(0,-1000,0.00001,0.00001), upper=c(1000,0.00001, 1000,1000))

# Print Results
estimate_par<-estimate$optim$bestmem
names(estimate_par)<-c("beta0","beta1", "sigmaM", "sigmaT")
estimate<-data.frame(cbind(estimate$optim$bestmem, c(beta0, beta1, sigmaM, sigmaT)))
colnames(estimate)<-c("estimate", "true")
rownames(estimate)<-c("beta0", "beta1", "sigmaM", "sigmaT")
estimate
```

```
##          estimate true
## beta0  180.068968  180
## beta1  -20.337769  -20
## sigmaM   5.128245    5
## sigmaT  10.416585   10
```

Python Code

```python
#works sometimes
import numpy as np
import math
import scipy.sparse as sp
import scipy.sparse.linalg as spln
from scipy.optimize import minimize
import scipy.stats as stats
import time


# Generate Data
```

```
N=1000; beta0 = 75; beta1 = -5; sigmaM = 5; sigmaT = 10
n=10 #size of matrix
delta=(np.diag(np.ones(n-1),-1)+np.diag(np.ones(n-1),1))*-sigmaM
#covariance matrix
Sigma=np.diag(np.ones(n))*(2*sigmaM**2+sigmaT**2)
Sigma=Sigma+delta
#mean vector
mu = np.zeros(n)
#simulation of MNV
x = np.random.random((N,))
y = beta0+beta1*x
y2= np.random.multivariate_normal(mu, Sigma, N)
y3=y2 + y[:,np.newaxis]


# Likelihood Function
def linMLE_MVN(params):
    beta0 = params[0]; beta1 = params[1]; sigmaM = params[2]; sigmaT = params[3]

    r1= x * beta1 + beta0
    r2 = y3 - r1[:,np.newaxis]
    delta=(np.diag(np.ones(n-1),-1)+np.diag(np.ones(n-1),1))*-sigmaM
    Sigma=np.diag(np.ones(n))*(2*sigmaM**2+sigmaT**2)
    sigma=Sigma+delta


    def MLE(x):
          aux1 = np.exp(-0.5*np.dot(np.dot((x),np.linalg.inv(sigma)),(x)))
          aux2 = np.sqrt(np.linalg.det(2*np.pi*sigma))
          mvn = aux1 / aux2
          return(mvn)

    loglik=sum(np.log(np.apply_along_axis(MLE, 1, r2)))


    return(-loglik)

# Optimization
initParams = [180, -20, 20, 20]
bnds = ((-1000,1000), (-1000, 1000), (0.01, 1000), (0.01, 1000))
results = minimize(linMLE_MVN, initParams,  method='SLSQP', bounds=bnds)


# Print Results
estimated_parameters = {'beta0':round(results.x[0],3), 'beta1':round(results.x[1],3),
'sigmaM':round(results.x[2],3), 'sigmaT':round(results.x[3],3)}
print 'estimate'
print estimated_parameters
parameters = {'beta0':beta0, 'beta1':beta1, 'sigmaM':sigmaM, 'sigmaT':sigmaT}
print 'true'
print parameters
```

```
## -c:39: RuntimeWarning: overflow encountered in exp
## -c:40: RuntimeWarning: invalid value encountered in sqrt
```

```
## estimate
## {'sigmaM': 6.372, 'sigmaT': 8.439, 'beta1': -5.184, 'beta0': 75.062}
## true
## {'sigmaM': 5, 'sigmaT': 10, 'beta1': -5, 'beta0': 75}
```