Guassian Proccess Project - Parameter Estimation

5/1/2022

```
library(MASS)
library(tinytex)
```

#Our project is to estimate the values of the hyperparameters parameters in a Guassian Process. Describe a Gaussian Process?

To estimate the hyper-parameters, we generated data as follows. y ~ N(0,

$$K_y$$

), where

$$K_y$$

is a semi-definite matrix. In this case we used

$$K_y = \sigma_f^2 e^{1/(2l^2)(X_i - X_j)^T(X_i - X_j)} + \sigma_y^2 I_n$$

, where

$$\sigma_y^2$$

controls the measurement noise,

$$\sigma_f^2$$

controls the vertical scale and

$$l^2$$

controls the length scale. The idea is that when values of x are close together, then the value of

$$f(x)$$

should also be close, where

$$f(x)$$

is a latent function which describes the relationship (without noise) between possble values of

$$x_i$$

and

$$y_i$$

.

#This section of code contains functions we need to use to either generate the data, or to estimate the hyper parameters.

```
s.d.2<-function(test){
  a<-length(test)
  diff.temp<-matrix(rep(NA,a*a),nrow=a,ncol=a)
  for (i in 1:a){ #This double loop gives (x_i-x_j)*(x_i-x-j) for
    for (j in 1:a){
      diff.temp[i,j]<-(test[i]-test[j])^2
    }
  }
  return(diff.temp)
}
#Vectorized
s.d<-function(test){
  X.mat<-(matrix(rep(test,length(test)),ncol=length(test),byrow=T))
  ###X.mat repeats X in each row.
  #Each column repeats the sample value of X. Column 1 has x1, col 2 has x2, ... The [i,j] cell has xj
  X.mat.new<-(matrix(rep(test,each=length(test)),ncol=length(test),byrow=T))
  #####X.mat.new repeats X in each column.
  #Each row repeats the same value of X. Row 1 has x1, row 2 has x2, ... The [i,j] cell has xi
  return((X.mat-X.mat.new)^2) #In the [i,j] cell will have (xi-xj)^2
}

#If x is D>1
s.d.D.2<-function(x){
  a<-nrow(x)        #Needed to know size of K.temp and for the loop.
  K.temp<-matrix(rep(NA,a*a),nrow=a,ncol=a)
  for (i in 1:a){   #This double loop gives t(X_i-X_j)*(X_i-X-j) for
    for (j in 1:a){       #each [i,j] cell in K.temp.
      K.temp[i,j]<-sum((x[i,]-x[j,])*(x[i,]-x[j,]))
    }
  }
  return(K.temp)
}
#Vectorized (D>1)
s.d.D<-function(x){
  D=ncol(x)          #Needed to set loop.
  temp<-list()       #Using a list so that can use the Reduce command.
  for (i in 1:D){    #Need to repeat for each column
    temp[[i]]<-s.d(x[,i]) #Calculates (x_ji-x_j'i)^2 for each column
  }
  return(Reduce('+',temp))   #This sums the corresponding cells.
}

#K written with inputs theta (The without noise component)
K.t<-function(x,t1,t2){
  D<-ncol(as.matrix(x))
  if (D==1){
  K.temp<-(exp(t1))*exp(-(1/(2*((exp(t2))))*s.d(x)))
  }
  if (D>1){
    K.temp<-(exp(t1))*exp(-(1/(2*((exp(t2))))*s.d.D(x)))
  }
  return(K.temp)
}
```

```
###My own Partition Function:
###This is used if we want to use Stochastic Descent.
create.batches<-function(Y,batch.size)
  {
  len<-length(Y)
  A<-c(1:len)
  temp<-sample(A,len,replace=FALSE)
  batchindex<-list()
  count<-len/batch.size
  for (i in 1:count){
    batchindex[[i]]<-temp[((i-1)*batch.size+1):((i)*batch.size)]
  }
  return(batchindex)
}
```

##The Log-Likelihood function given by Murphy [2012] is :

$$log(p(y|X)) = -\frac{1}{2}y'K_y^{-1}y - log(|K_y|) - \frac{N}{2}log(2\pi)$$

Here is the function to compute the negative log likelihood:

```
#Negative log likelihood written as a function of thetas
nLL.t<-function(x,Y,t1,t2,t3){
  K.temp<-K.t(x,t1,t2)+(exp(t3))*diag(length(Y))
  L<-t(chol(K.temp))  #K = LL'
  L.inv<-solve(L)
  alpha<-t(L.inv)%*%L.inv%*%Y #alpha = (L')^-1*L^-1*Y = K^-1 *Y
  N<-length(Y)
  log.like<-(-1/2*as.numeric(t(Y)%*%alpha)-(sum(log(diag(L))))-N/2*log(2*pi))
  #log.like<-(-1/2*as.numeric(t(Y)%*%alpha)-(sum(diag(log(L))))-N/2*log(2*pi))
  return(-1*log.like)
}
```

The derivative of the Log-Likelihood is then:

$$\frac{\partial log(p(y|X))}{\partial \theta_j} = \frac{1}{2}y^T K_y^{-1}\frac{\partial K_y}{\partial \theta_j}K_y^{-1}y - \frac{1}{2}tr(K_y^{-1}\frac{\partial K_y}{\partial \theta_j})$$

For clarity of notation let:

$$\alpha = K_y^{-1}y$$

then the derivative of the log likelihood is:

$$\frac{1}{2}tr((\alpha\alpha^T - K_y^{-1})\frac{\partial K_y}{\partial \theta_j})$$

where

$$\frac{\partial K_y}{\partial \theta_j}$$

is the partial derivative of

$$K_y$$

with respect to parameter

$$\theta_j : j = 1, 2, 3.$$

These partials for the squared-exponential kernel are derived below.

#Derivatives w.r.t theta The squared-exponential kernel is:

$$K_y = \sigma_f^2 * e^{-\frac{1}{2l^2} s.d(X)} + \sigma_y^2 I_n$$

To address the fact that

$$\sigma_f^2, l^2, \sigma_y^2$$

must be positive, we made the substitution recommended by Murphy [2012] of

$$\theta_1 = log(\sigma_f^2), \theta_2 = log(l^2), \theta_3 = log(\sigma_y^2)$$

Then the squared exponential kernel in terms of

$$\theta_j$$

is

$$K_y = e^{\theta_1} * e^{-\frac{1}{2e^{\theta_2}} s.d(X)} + e^{\theta_3} I_n$$

Let s.d(X) for an nxn X matrix be the squared distance function given by:

$$s.d(X) = (X_i - X_j)^T (X_i - X_j)$$

Then the partial derivatives of

$$K_y$$

are

$$\frac{\partial K_y}{\partial \theta_1} = e^{\theta_1} e^{-\frac{1}{2e^{\theta_2}} s.d(X)}$$

$$\frac{\partial K_y}{\partial \theta_2} = -\frac{1}{2} e^{\theta_1} (-1 e^{-\theta_2} s.d(X)) e^{-\frac{1}{2} e^{-\theta_2} s.d(X)}$$

Note: The derivative is defined on each [i,j] cell, and the multiplication indicated by the multiplication above is not matrix multiplication but multiplication of corresponding cells. In other words: For cell [i,j]:

$$[\frac{\partial K_y}{\partial \theta_2}]_{[i,j]} = -\frac{1}{2} e^{\theta_1} (-1 e^{-\theta_2} (X_i - X_j)^T (X_i - X_j)) e^{-\frac{1}{2} e^{-\theta_2} (X_i - X_j)^T (X_i - X_j)}$$

$$\frac{\partial K_y}{\partial \theta_3} = e^{\theta_3} I_n$$

Here are the functions used to calculate the gradient. The first 3 functions calculate the partial derivative of the kernel with respect to each

$$\theta$$

. The third function combines the specified partial with the common matrix in each of the 3 partials for the log likelihood (

$$\alpha^T \alpha - K_y^{-1}$$

where

$$\alpha = K_y^{-1} y$$

) as was derived in Murphy. The last function combines the first four functions to input the data and current parameters and output the gradient as a vector.

```
der.theta1<-function(x,theta1,theta2){
  D=(ncol(as.matrix(x)))
  if (D==1){
  K.temp<-(exp(theta1))*exp(-(1/(2*(exp(theta2))))*(s.d(x)))
  }
  if (D>1){
    K.temp<-(exp(theta1))*exp(-(1/(2*(exp(theta2))))*(s.d.D(x)))
  }
  return(K.temp)
}
der.theta2<-function(x,theta1,theta2){
  D=(ncol(as.matrix(x)))
  if (D==1){sd<-s.d(x)}
  if (D>1){sd<-s.d.D(x)}
  K.temp<-((-1/2)*((-1*exp(-1*theta2)))*(exp(theta1))*sd*exp(-(1/2)*((exp(-1*theta2)))*sd))
  return(K.temp)
}
der.theta3<-function(t3,len){
  temp<-exp(t3)*diag(len)
  return(temp)
}
gradient<-function(K,partial,y){
  len<-length(y)
  L<-t(chol(K))
  K.inv<-solve(t(L))%*%solve(L)
  alpha<-K.inv%*%as.matrix(y,nrow=len,ncol=1)
  temp<-0.5*sum(diag((alpha%*%t(alpha)-K.inv)%*%partial))
  return(temp)
}
gr.v<-function(Y,x,t1,t2,t3){
  len<-length(Y)
  K.temp<-K.t(x,t1,t2)+exp(t3)*diag(length(Y))
  gr1<-der.theta1(x,t1,t2)
  gr2<-der.theta2(x,t1,t2)
  gr3<-der.theta3(t3,len)
  return(c(gradient(K.temp,gr1,Y),gradient(K.temp,gr2,Y),gradient(K.temp,gr3,Y)))
}
```

The first two functions in the following code compute a single update of beta in the case of momentum (function 1) or gradient descent(function 2). The third function is the iterative function is to compute gradient descent (with or without momentum and with or without stochastic). The inputs are as follows: 1-5) x, Y, t1, t1, t3 (the data and the current values of the

$$\theta$$

's). 6) iter.number = number of iterations (or number of epochs in the case of stochastic descent). 7) lrat=Learning rate vector. 8) If want to select stochastic option. Enter 1 if want stochastic. If do not want stochastic then can leave blank or put 0. 9) num.batch = number of batches for each epoch. If not stochastic can leave blank or enter 1. 10) If want to select momentum as an option enter 1. If do want momentum can leave blank or enter 0. 11) gamma = a coefficient (between 0 and 1) which gives how much the previous value of v influences the new value of v. Values closer to 1 gives the previous value more influence. A value for gamma

```
#momentum update function (used in larger function)
momentum.update<-function(x,Y,biter,v,gamma,lrat)
{
        gv = gr.v(Y,x,biter[1],biter[2],biter[3])
        v.new= gamma*v - lrat*(gv)
        biter = biter - v.new
  return(list(biter,v.new,gv))
      }

#Gradient Descent Update function (used in larger function)
grad.desc.update<-function(x,Y,biter,lrat){
  gr<-gr.v(Y,x,biter[1],biter[2],biter[3])
   biter = biter + lrat*(gr)
   return(list(biter,gr))
}

###Iterative Function
theta.grad<-function(x,Y,theta1.0,theta2.0,theta3.0,iter.number,lrat,
                     stochastic=NULL,num.batch=NULL,momentum=NULL,gamma=NULL){
D<-ncol(as.matrix(x))
if (is.null(num.batch)){num.batch<-1}
#Initialize
len<-length(Y)
biter<-c(theta1.0,theta2.0,theta3.0)
niter<-iter.number # number of epochs
#Storage
bM2 = matrix(0.0,niter*num.batch+1,length(biter))
bM2[1,] = biter
lvv<-c(rep(NA,niter*num.batch+1))
lvv[1]<-nLL.t(x,Y,biter[1],biter[2],biter[3])
if (is.null(stochastic)){stochastic<-0}
stochastic<-stochastic
partition<-list()
if (stochastic==0){partition[[1]]<-c(1:len)}
if (is.null(momentum)){momentum<-0}
momentum<-momentum
if (momentum==1){
  v<-matrix(NA,nrow=niter*num.batch+1,ncol=length(biter))
  v[1,]<-c(rep(0,length(biter)))
  }
## iterate
for(i in 1:niter) {
  if (stochastic==1){  #If stochastic select partition
    batch.size=length(Y)/num.batch
partition<-create.batches(Y,batch.size)
    }
  for (j in 1:num.batch){
    D<-ncol(as.matrix(x))
  selection<-partition[[j]]
  if (momentum==0){
    if (D==1){
   out<-grad.desc.update(x[selection],Y[selection],biter,lrat[num.batch*(i-1)+j])
    biter<-out[[1]]
```

```
    gr<-out[[2]]
    }
    if (D>1){
      out<-grad.desc.update(x[selection,],Y[selection],biter,lrat[num.batch*(i-1)+j])
    biter<-out[[1]]
    gr<-out[[2]]
    }
   }
  if (momentum==1){
  if (D==1){
  out<-momentum.update(x[selection],Y[selection],biter,v[num.batch*(i-1)+j,]
                       ,gamma,lrat[num.batch*(i-1)+j])
  biter<-out[[1]]
  v[num.batch*(i-1)+j+1,]<-out[[2]]
  gr<-out[[3]]
  }
  if (D>1){
    out<-momentum.update(x[selection,],Y[selection],biter,v[num.batch*(i-1)+j,]
                         ,gamma,lrat[num.batch*(i-1)+j])
  biter<-out[[1]]
  v[num.batch*(i-1)+j+1,]<-out[[2]]
  gr<-out[[3]]
  }
  }
   lvv[num.batch*(i-1)+j+1]<-nLL.t(x,Y,biter[1],biter[2],biter[3])
   bM2[num.batch*(i-1)+j+1,]<-biter
  }
  }
return(list(bM2,gr,lvv))
}
```

Function to plot contours This code is so that we do not need to repeat it for each contour we draw.

```
###Contour Plot Function
contour.plot.GP<-function(gs,a,b,c,d,fixed.theta,biter, x,Y){
alga1 = seq(from=a,to=b,length.out=gs) #one d grid
alga2 = seq(from=c,to=d,length.out=gs) #one d grid
alg2 = expand.grid(alga1,alga2) # two d grid
llv = rep(0,nrow(alg2))
quadv = rep(0,nrow(alg2))
linv = rep(0,nrow(alg2))
if(fixed.theta==1){
for(i in 1:length(llv)) {
   llv[i] = nLL.t(x,Y,biter[1],(alg2[i,1]),(alg2[i,2]))
}
llmat = matrix(as.numeric(llv),nrow=length(alga1),ncol=length(alga2))
}
if(fixed.theta==2){
for(i in 1:length(llv)) {
   llv[i] = nLL.t(x,Y,(alg2[i,1]),biter[2],(alg2[i,2]))
}
llmat = matrix(as.numeric(llv),nrow=length(alga1),ncol=length(alga2))
}
```

```
if(fixed.theta==3){
for(i in 1:length(llv)) {
   llv[i] = nLL.t(x,Y,(alg2[i,1]),(alg2[i,2]),biter[3])
}
llmat = matrix(as.numeric(llv),nrow=length(alga1),ncol=length(alga2))
}
return(list(alga1,alga2,llmat))
}
```

Function to determine where to stop drawing arrows to avoid error messages. bM=Two column matrix recording the draws for each iteration. a=cut-off for arrow size (in one direction).

```
stop.num<-function(bMo,a){
i<-1
while ((abs(bMo[i+1,1]-bMo[i,1])>=a) & ((i+1)<nrow(bMo))){i<-i+1}
j<-1
while ((abs(bMo[j+1,2]-bMo[j,2])>=a) & ((j+1)<nrow(bMo))){j<-j+1}
return(c(i,j))
}
```

#In this section we generate the data in the case where there is only 1 x variable (D=1).

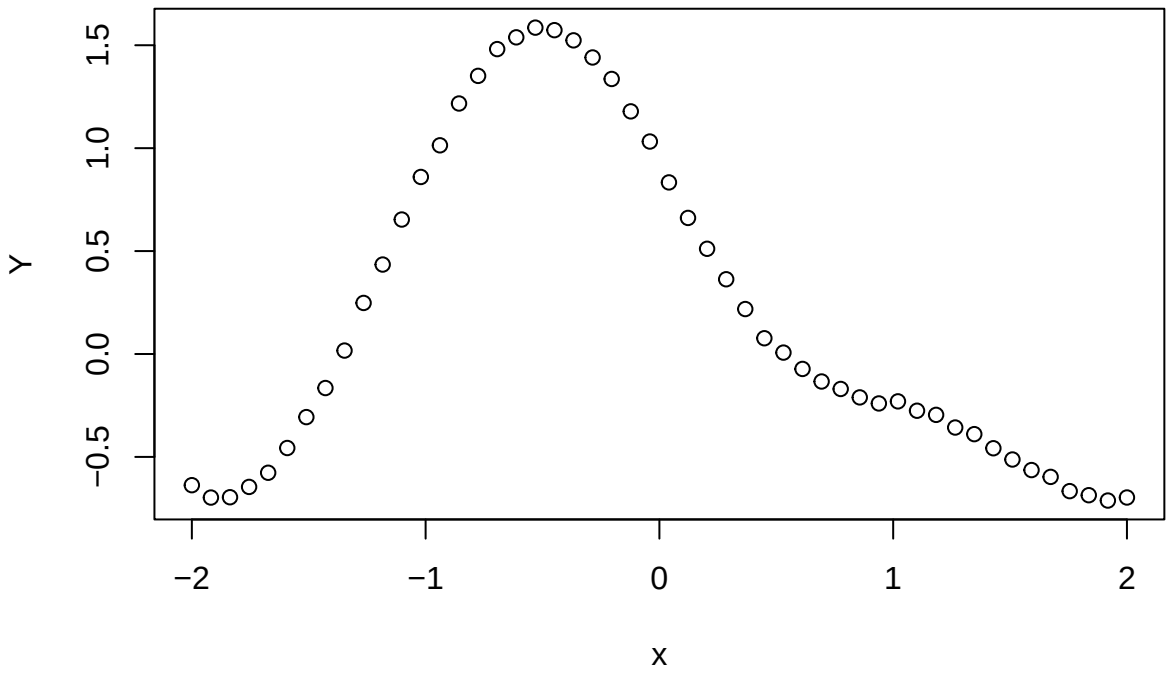
##Generate the Data (D=1) ###Set Parameters

```
sig.f.true<-0.5
l.true<-0.7
sig.y.true<-.01
t1<-theta1.true<-log(sig.f.true^2)
t2<-theta2.true<-log(l.true^2)
t3<-theta3.true<-log(sig.y.true^2)
beta.true2<-c(sig.f.true,l.true)
theta.true2<-c(t1,t2)
beta.true3<-c(sig.f.true,l.true,sig.y.true)
```

###Generate Data Y~N(0,K_y), K_y = k(x_i,x_j)

```
set.seed(5312)
#Generate x
n=50
x<-seq(-2,2,length.out=n)
x.save<-x
#x<-rnorm(n,0,1)

#Generate Covariance Matrix
length.x<-length(x)
K.y<-K.t(x,t1,t2)+exp(t3)*diag(1,length.x)

#Generate Y~N(0,Ky)
ev<-eigen(K.y)
L.e<-ev$values
P<-ev$vectors
D.5 = diag(sqrt(L.e))
A = P%*%D.5
```

```r
mu<-c(rep(0,length.x)) #Zero Mean
Z<-rnorm(length.x,mean=0,sd=1)
Y<-mu+A%*%Z
y.save<-Y #To use at code in the end.
plot(x,Y)
```

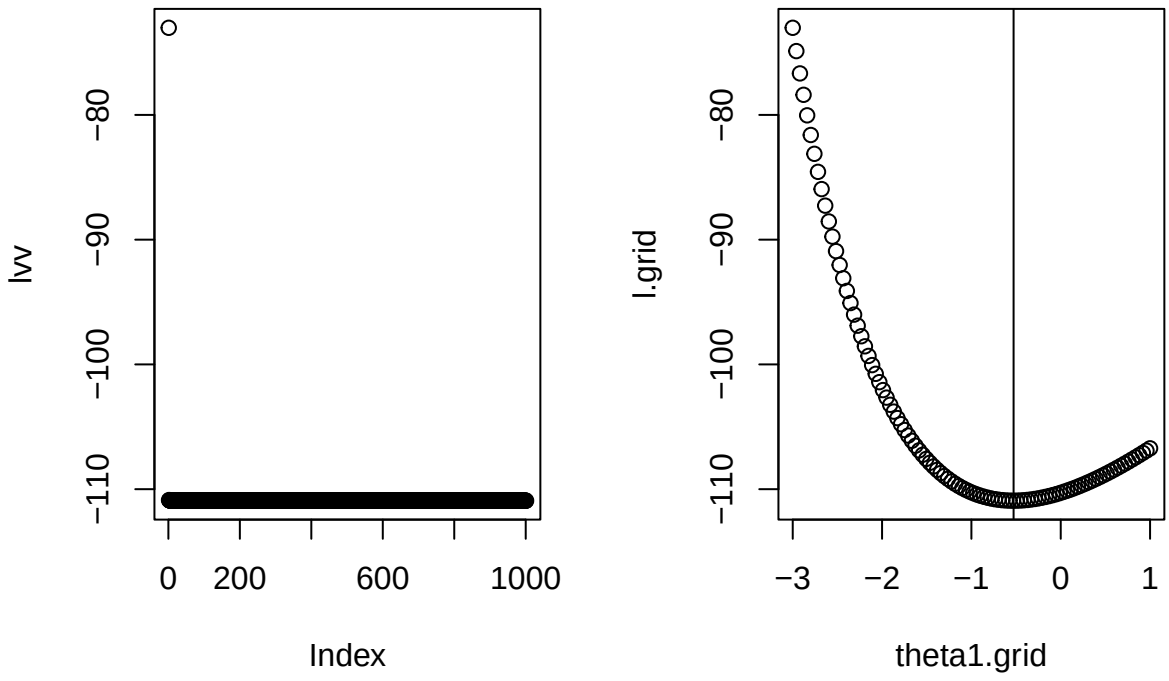


#First Investigate Gradient Descent with 2 fixed parameters. Look at llv vs theta.grid #Save min(llv) #Maybe Look at arrows on the grid.

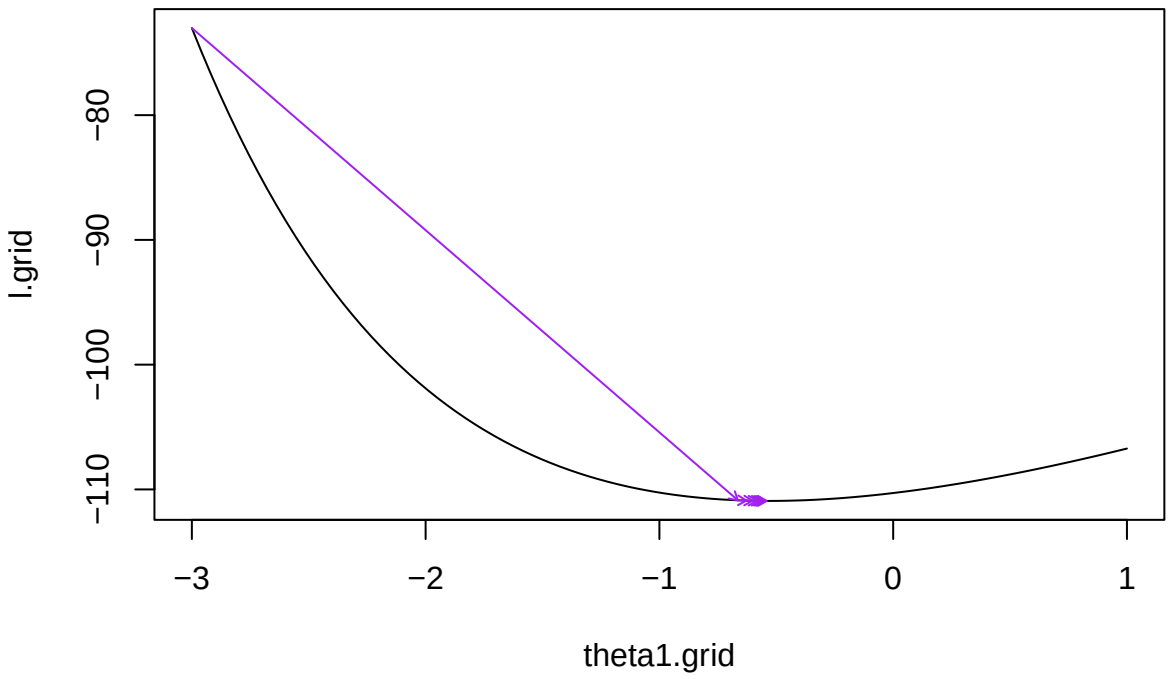
Two fixed theta values. Gradient descent with the free one.

```r
#Initialize
theta1.0=-3
theta2.0=-0.5
theta3.0=-11
biter=c(0.5)
niter<-1000 # number of iterations
lrat = rep(.05,niter) #learning rate
for (t in 1:niter){
  lrat[t]<-lrat[1]/(1+.01*t)
}

#Storage
bM2 = matrix(0.0,niter+1,length(biter))
lvv<-c(rep(NA,niter+1))
```

```
par(mfrow=c(1,2))
##Investigating Theta1
biter<-c(theta1.0)
lvv[1]<-nLL.t(x,Y,biter[1],t2,t3)
bM2[1] = biter

## iterate
for(i in 1:niter) {
   gr<-gr.v(Y,x,biter[1],t2,t3)
   biter = biter + lrat[i]*(gr[1])
   lvv[i+1]<-nLL.t(x,Y,biter,t2,t3)
   bM2[i+1]<-biter
  }
b<-biter

plot(lvv)
theta1.grid<-seq(-3,1,length.out=100)
l.grid<-c(rep(NA,100))
for (i in 1:100){
l.grid[i]<-nLL.t(x,Y,theta1.grid[i],t2,t3)
}
plot(theta1.grid,l.grid)
abline(v=b)
```

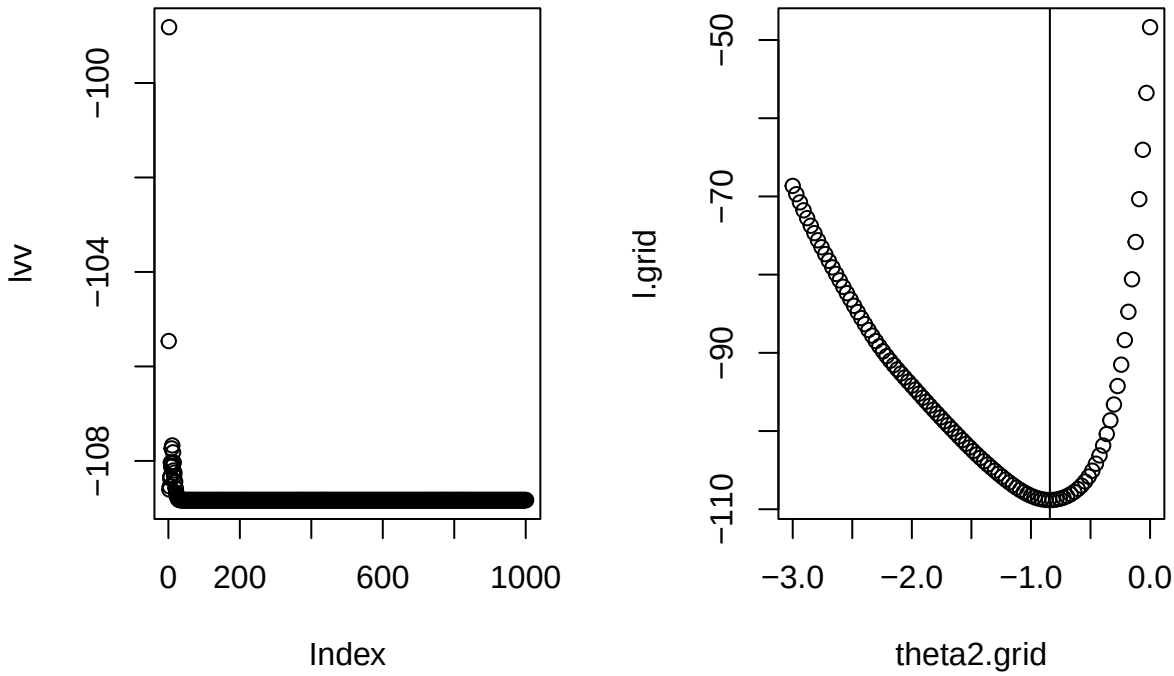


```
par(mfrow=c(1,1))
plot(theta1.grid,l.grid, type='l')
for(i in 2:(10)) {
   arrows(bM2[i-1],lvv[i-1],bM2[i],lvv[i],length=.05,col="purple")}
```

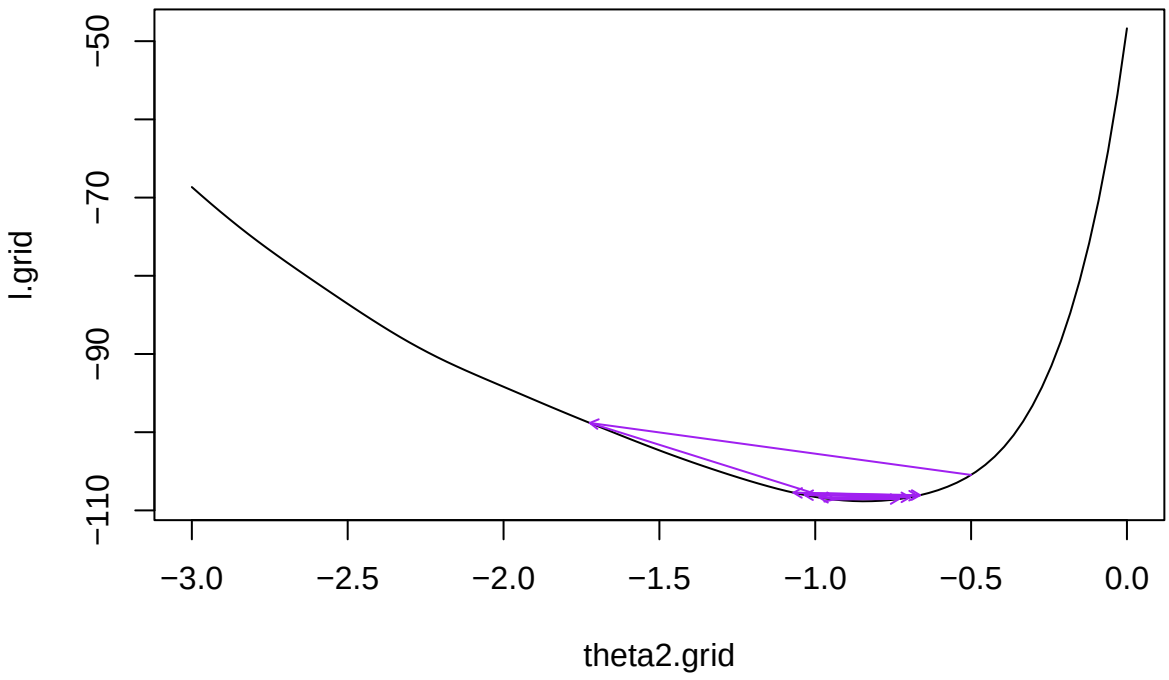



```
##Investigating Theta2
biter<-c(theta2.0)
lvv[1]<-nLL.t(x,Y,t1,biter,t3)
bM2[1] = biter

## iterate
for(i in 1:niter) {
   gr<-gr.v(Y,x,t1,biter[1],t3)
   biter = biter + lrat[i]*(gr[2])
   lvv[i+1]<-nLL.t(x,Y,t1,biter,t3)
   bM2[i+1]<-biter
  }
b<-biter

par(mfrow=c(1,2))
plot(lvv)
theta2.grid<-seq(-3,0,length.out=100)
l.grid<-c(rep(NA,100))
for (i in 1:100){
l.grid[i]<-nLL.t(x,Y,t1,theta2.grid[i],t3)
```

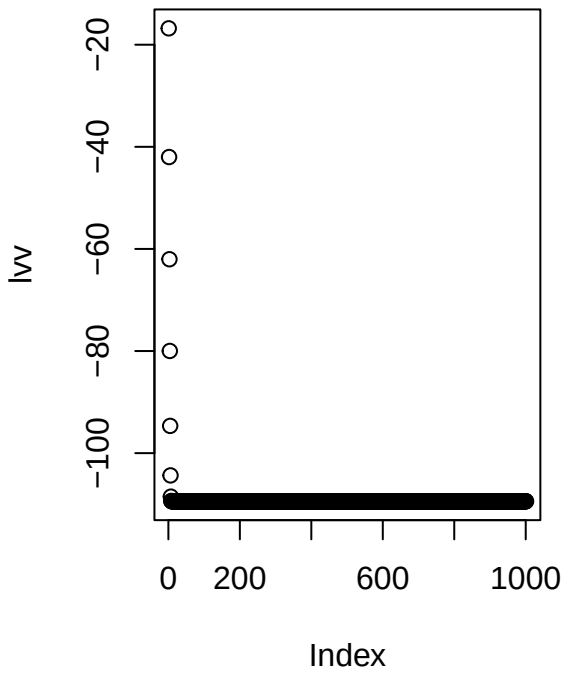
```
}
plot(theta2.grid,l.grid)
abline(v=b)
```

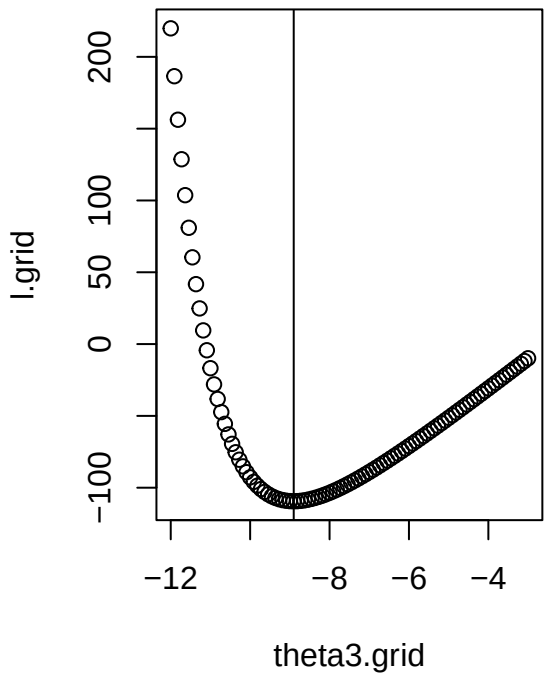



```
par(mfrow=c(1,1))
plot(theta2.grid,l.grid, type='l')
for(i in 2:(10)) {
   arrows(bM2[i-1],lvv[i-1],bM2[i],lvv[i],length=.05,col="purple")}
```

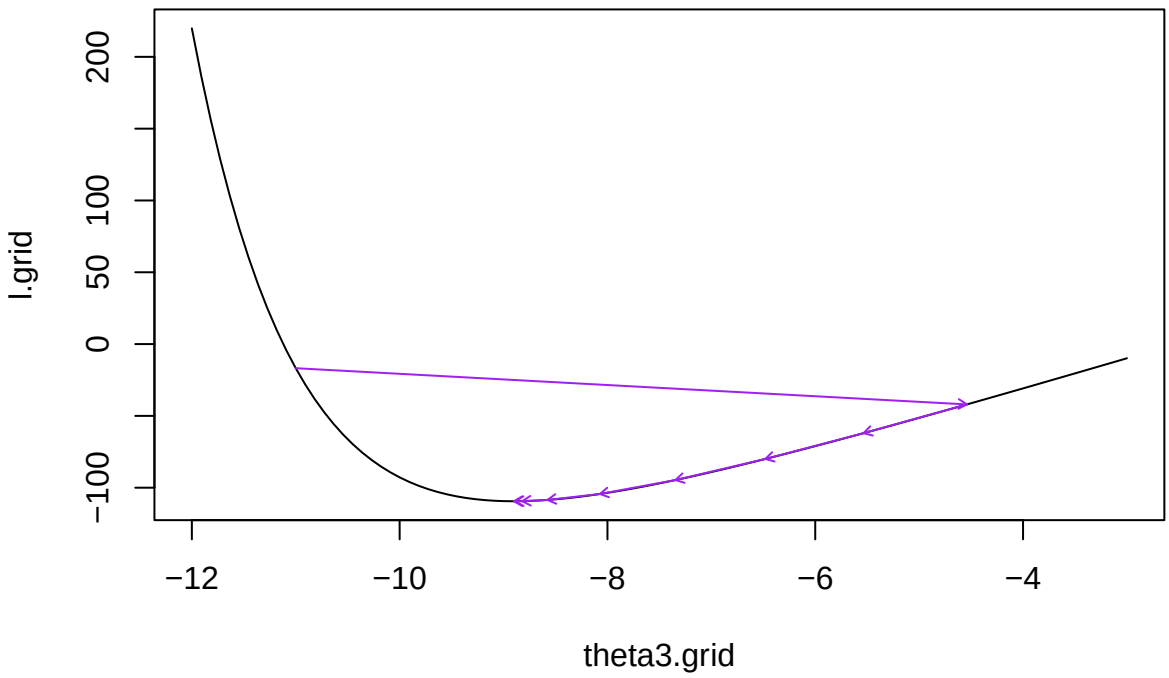



```
##Investigating Theta3
biter<-c(theta3.0)
lvv[1]<-nLL.t(x,Y,t1,t2,biter)
bM2[1] = biter

## iterate
for(i in 1:niter) {
   gr<-gr.v(Y,x,t1,t2,biter[1])
   biter = biter + lrat[i]*(gr[3])
   lvv[i+1]<-nLL.t(x,Y,t1,t2,biter[1])
   bM2[i+1]<-biter
  }
b<-biter

par(mfrow=c(1,2))
plot(lvv)
theta3.grid<-seq(-12,-3,length.out=100)
l.grid<-c(rep(NA,100))
for (i in 1:100){
l.grid[i]<-nLL.t(x,Y,t1,t2,theta3.grid[i])
}
plot(theta3.grid,l.grid)
abline(v=b)
```






```
par(mfrow=c(1,1))
plot(theta3.grid,l.grid, type='l')
for(i in 2:(10)) {
   arrows(bM2[i-1],lvv[i-1],bM2[i],lvv[i],length=.05,col="purple")}
```



If we know two of the paramaters, gradient descent converged quickly for the other parameter. It looks like our functions are working properly.

#Gradient Descent when we know 1 hyper parameter.

```
#Initialize
theta1.0=-2
theta2.0=-1.5
theta3.0=-5
niter<-2000 # number of iterations
lrat = rep(.005,niter) #learning rate


##Fix t3
#Storage
biter<-c(theta1.0,theta2.0,theta3.0)
bM2 = matrix(0.0,niter+1,length(biter[1:2]))
bM2[1,] = biter[1:2]
lvv<-c(rep(NA,niter+1))
lvv[1]<-nLL.t(x,Y,biter[1],biter[2],t3)
#K.y.temp<-K.y
## iterate
for(i in 1:niter) {
   gr<-gr.v(Y,x,biter[1],biter[2],t3)
   biter[1:2] = biter[1:2] + lrat[i]*(gr[1:2])
   lvv[i+1]<-nLL.t(x,Y,biter[1],biter[2],t3)
   bM2[i+1,]<-biter[1:2]
```

```
  }
biter
```

```
## [1]  0.06198403 -0.44711399 -5.00000000
```

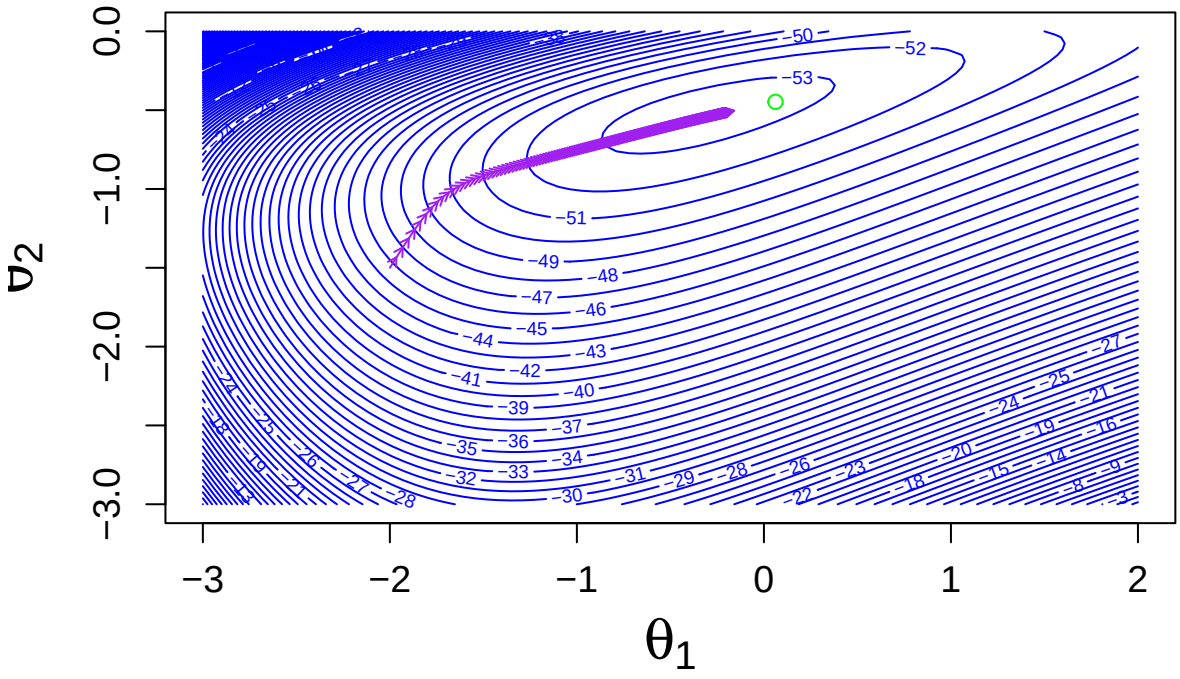
```
gr
```

```
## [1] 1.945057e-07 4.669571e-08 6.160682e+00
```

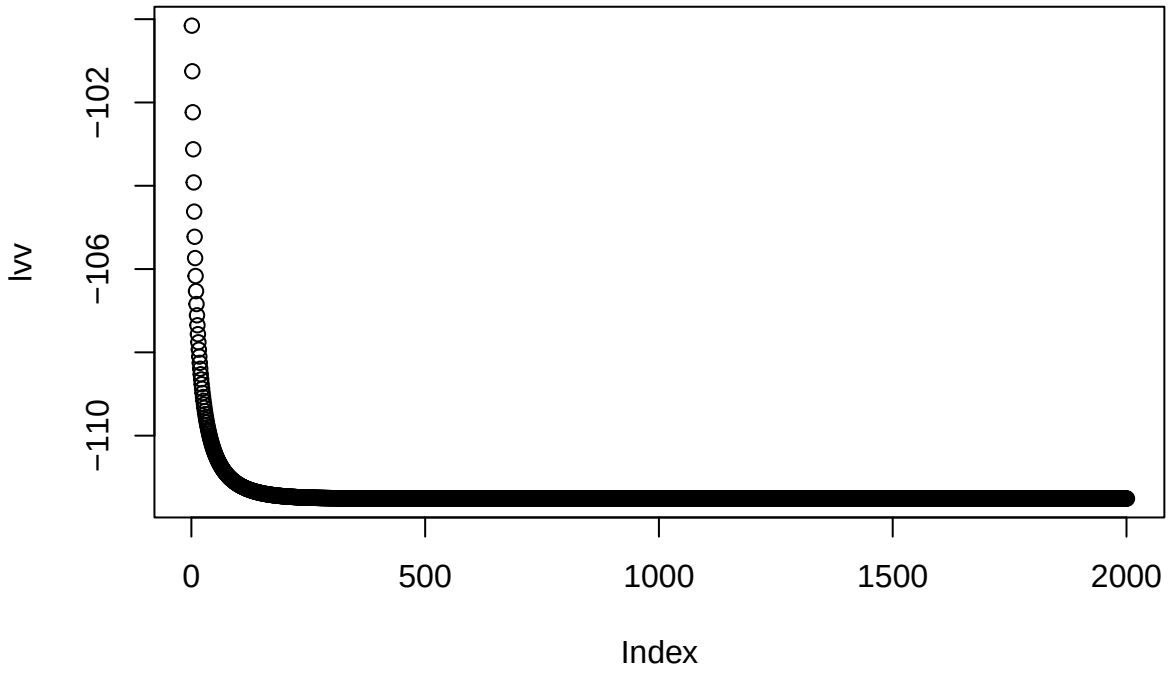
```
lvv[niter+1]
```

```
## [1] -111.5105
```

```
out<-contour.plot.GP(80,-3,2,-3,0,3,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=80,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[2]),col="blue",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
bM3<-bM2[,1:2]
st<-stop.num(bM3,.0005)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col="purple")}
points(bM3[niter+1,1],bM3[niter+1,2],col="green")
```



```
plot(lvv)
```




```
#####Fix t2
biter<-c(theta1.0,theta2.0,theta3.0)
bM2 = matrix(0.0,niter+1,length(biter[1:2]))
bM2[1,] = biter[c(1,3)]
lvv<-c(rep(NA,niter+1))
lvv[1]<-nLL.t(x,Y,biter[1],t2,biter[3])
#K.y.temp<-K.y
## iterate
for(i in 1:niter) {
   gr<-gr.v(Y,x,biter[1],t2,biter[3])
   biter[c(1,3)] = biter[c(1,3)] + lrat[i]*(gr[c(1,3)])
   lvv[i+1]<-nLL.t(x,Y,biter[1],t2,biter[3])
   bM2[i+1,]<-biter[c(1,3)]
  }
biter
```

```
## [1] -0.5500705 -1.5000000 -8.9318894
```

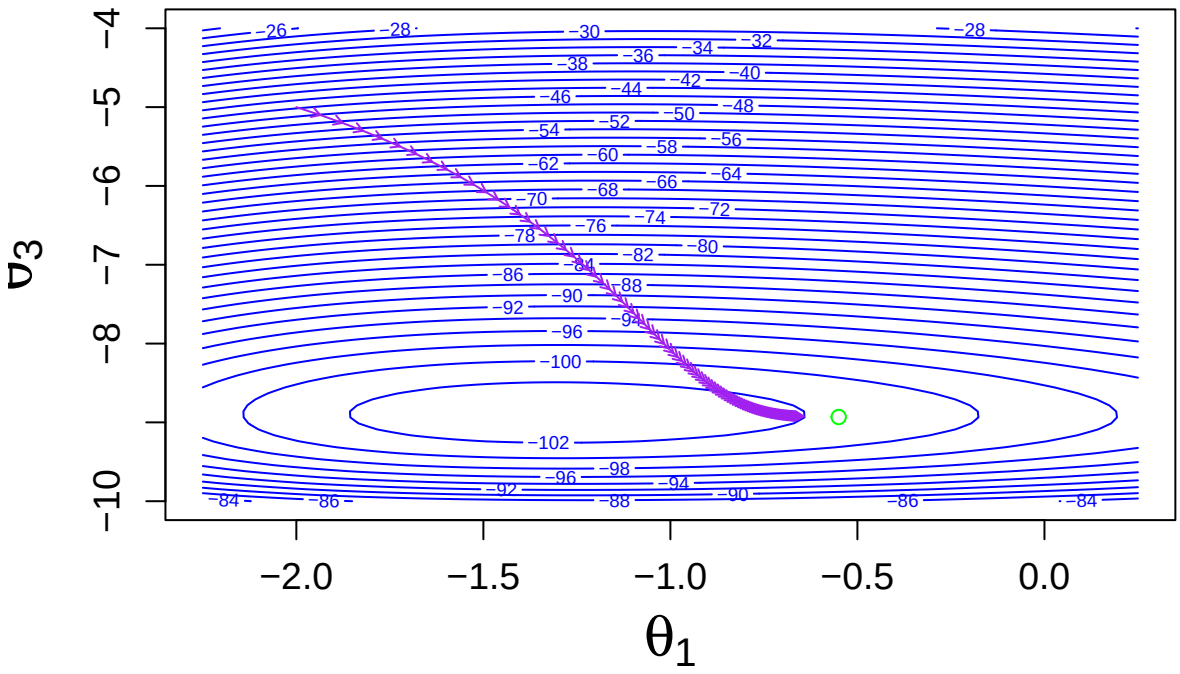
```
gr
```

```
## [1] -4.418022e-12  4.346694e+00 -1.006686e-11
```

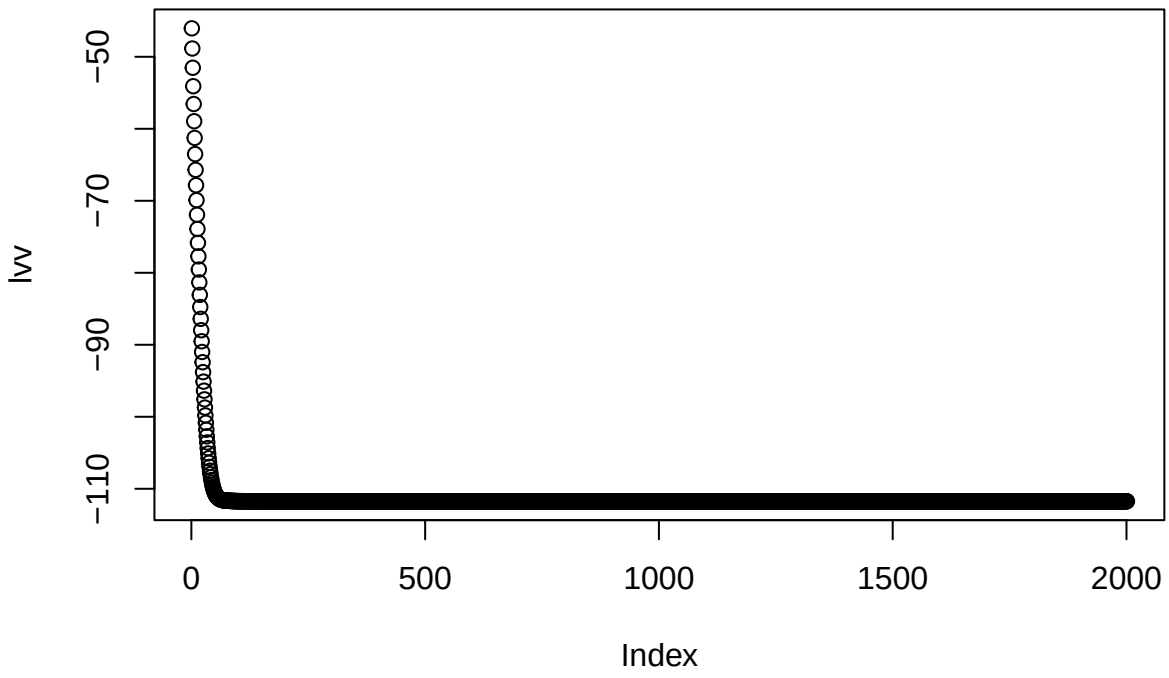
```
lvv[niter+1]
```

```
## [1] -111.7324
```

```
out<-contour.plot.GP(80,-2.25,0.25,-10,-4,2,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=40,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[3]),col="blue",
           cex.lab=1.8,cex.axis=1.2,lwd=1)
bM4<-bM2[,1:2]
st<-stop.num(bM4,.0005)
for(i in 2:(min(st))) {
   arrows(bM4[i-1,1],bM4[i-1,2],bM4[i,1],bM4[i,2],length=.05,col="purple")}
points(bM4[niter+1,1],bM4[niter+1,2],col="green")
```




```
plot(lvv)
```




```
#####Fix t1
biter<-c(theta1.0,theta2.0,theta3.0)
bM2 = matrix(0.0,niter+1,length(biter[1:2]))
bM2[1,] = biter[2:3]
lvv<-c(rep(NA,niter+1))
lvv[1]<-nLL.t(x,Y,t1,biter[2],biter[3])
#K.y.temp<-K.y
## iterate
for(i in 1:niter) {
   gr<-gr.v(Y,x,t1,biter[2],biter[3])
   biter[c(2,3)] = biter[c(2,3)] + lrat[i]*(gr[c(2,3)])
   lvv[i+1]<-nLL.t(x,Y,t1,biter[2],biter[3])
   bM2[i+1,]<-biter[c(2,3)]
  }
biter
```

```
## [1] -2.0000000 -0.8314411 -8.9173350
```

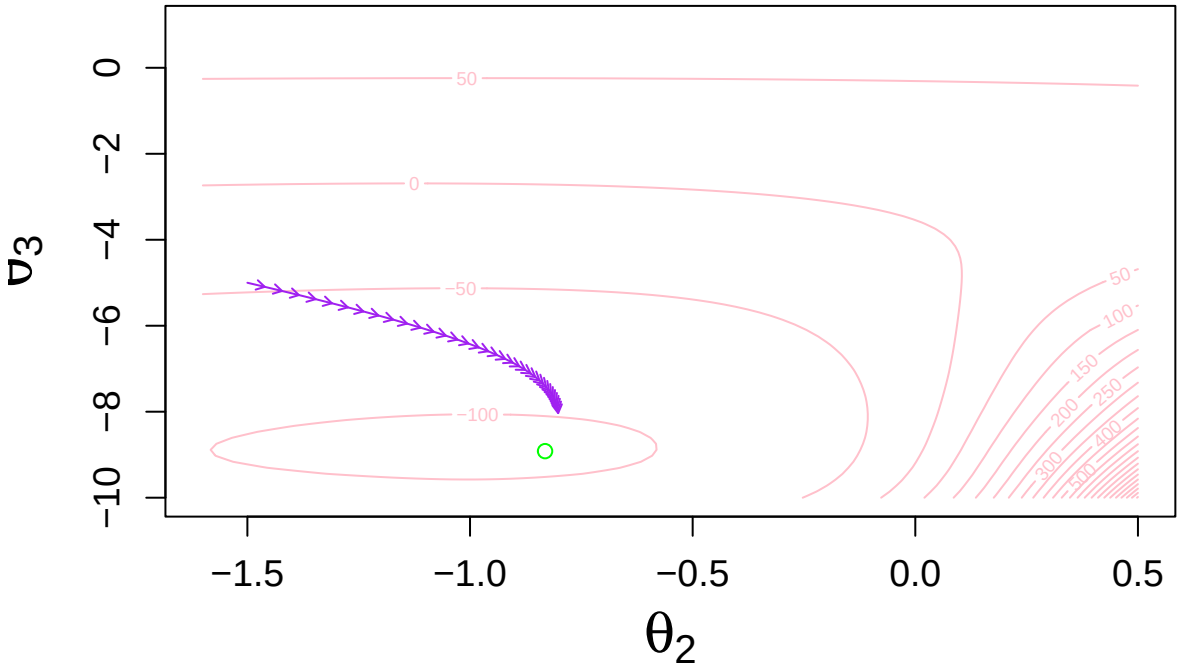
```
gr
```

```
## [1]  4.748915e+00  4.859113e-12 -5.615577e-12
```

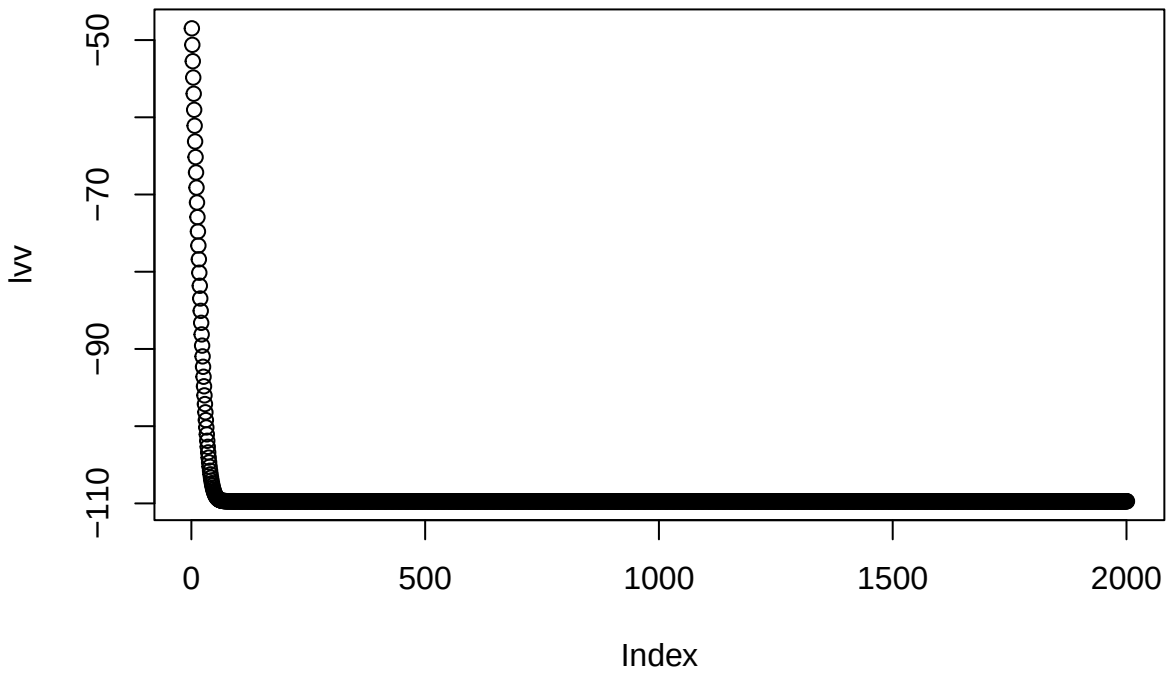
```
lvv[niter+1]
```

```
## [1] -109.7199
```

```
out<-contour.plot.GP(80,-1.6,0.5,-10,1,1,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=40,drawlabels=TRUE,
     xlab=expression(theta[2]),ylab=expression(theta[3]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
bM5<-bM2[,1:2]
st<-stop.num(bM5,.0005)
for(i in 2:(min(st))) {
   arrows(bM5[i-1,1],bM5[i-1,2],bM5[i,1],bM5[i,2],length=.05,col="purple")}
points(bM5[niter+1,1],bM5[niter+1,2],col="green")
```




```
plot(lvv)
```



Fixing one of the thetas at the true value of theta, we notice that the log-likelihood is not maximized at the true values of theta (since the gradient is still not close to 0 for the fixed parameter). Looking at the contour plot where

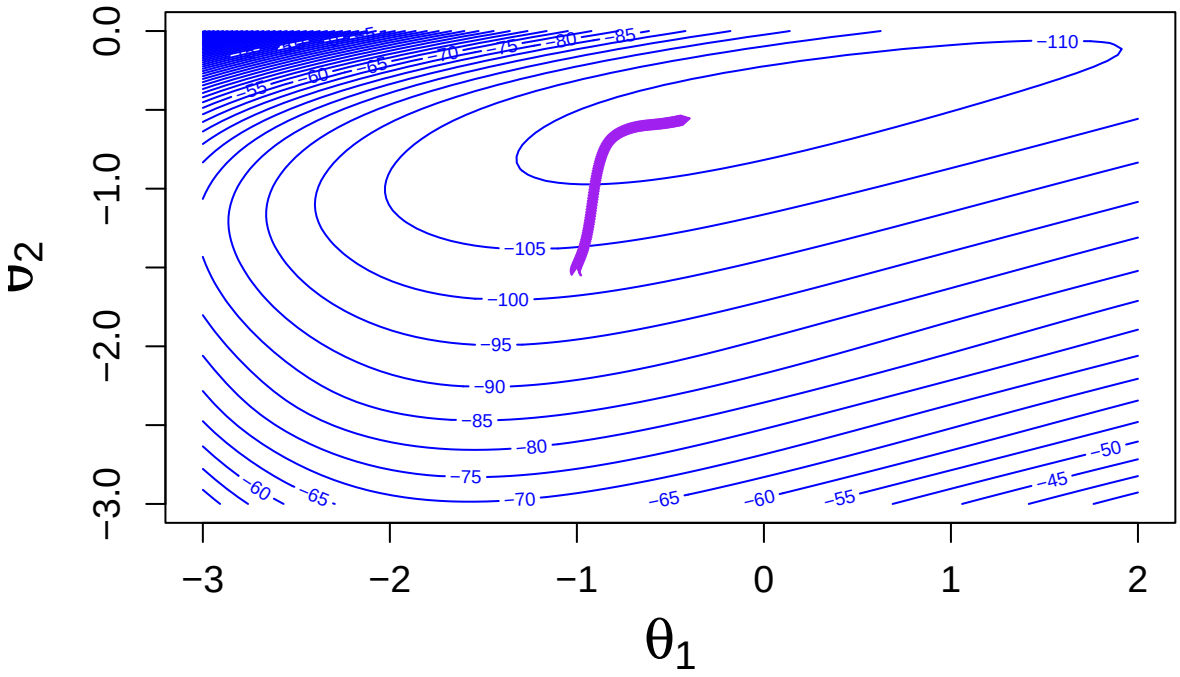
$$\theta_2$$

is fixed, we see that it led to gradient descent converging outside the trough.

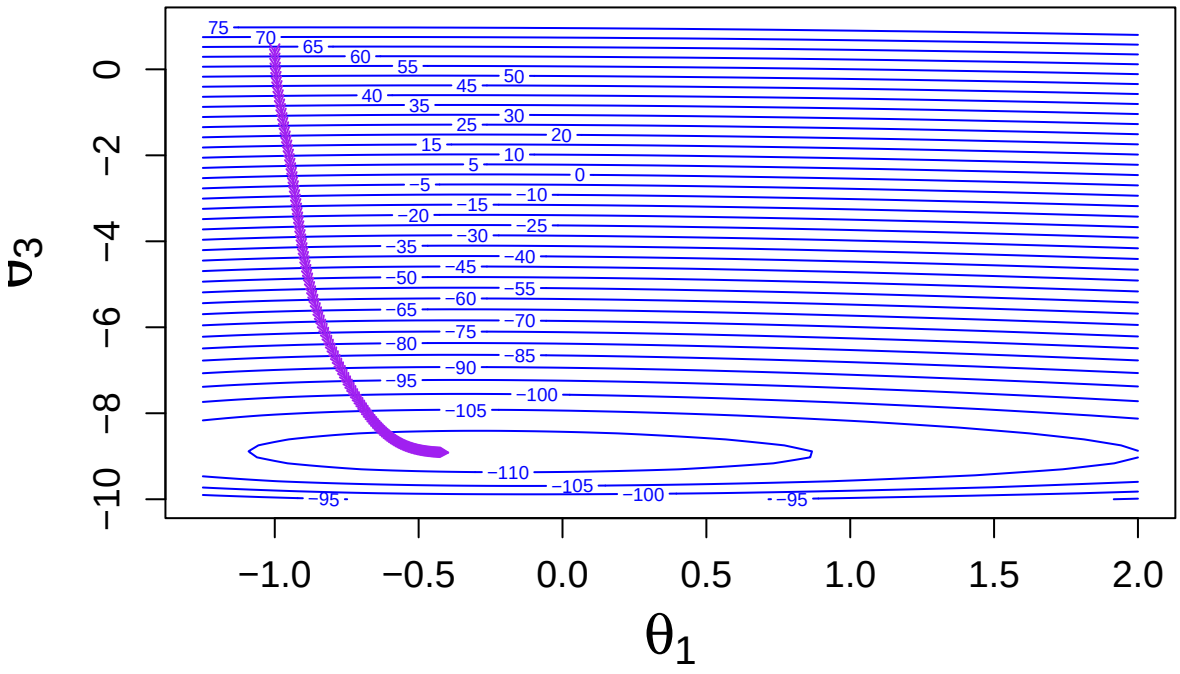
#Gradient Descent for all 3 hyper-parameters. Updating All three

```
set.seed(1234)
theta1.0=-1
theta2.0=-1.5
theta3.0=0.5
niter<-140
num.batch<-1
lrat = rep(.005,niter) #learning rate
# lrat[1]=.005
# for (t in 2:(niter)){
# lrat[t] = lrat[1]/(1+0.1*t) #learning rate
# }
result<-theta.grad(x,Y,theta1.0,theta2.0,theta3.0,niter,lrat)
bM2<-result[[1]]
gr<-result[[2]]
lvv<-result[[3]]
biter<-bM2[num.batch*niter+1,]
######Contour Plot theta3=biter[3]
out<-contour.plot.GP(80,-3,2,-3,0,3,biter,x,Y)
```

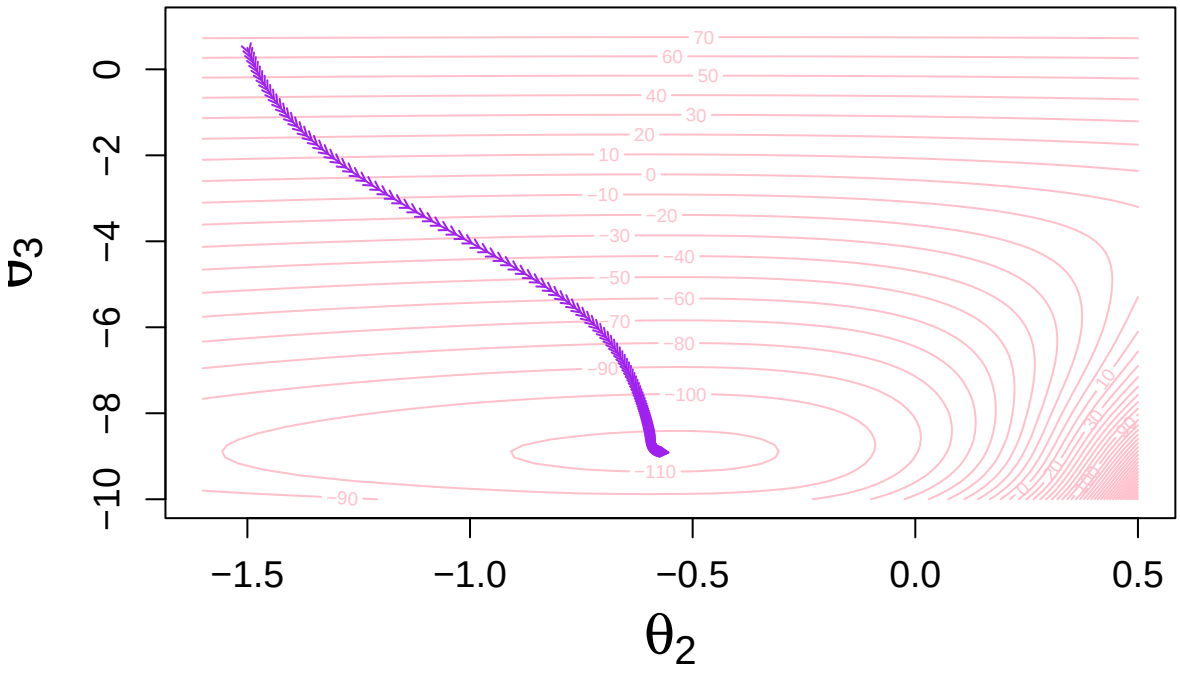
```
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=80,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[2]),col="blue",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
bM3<-bM2[,1:2]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col="purple")}
```



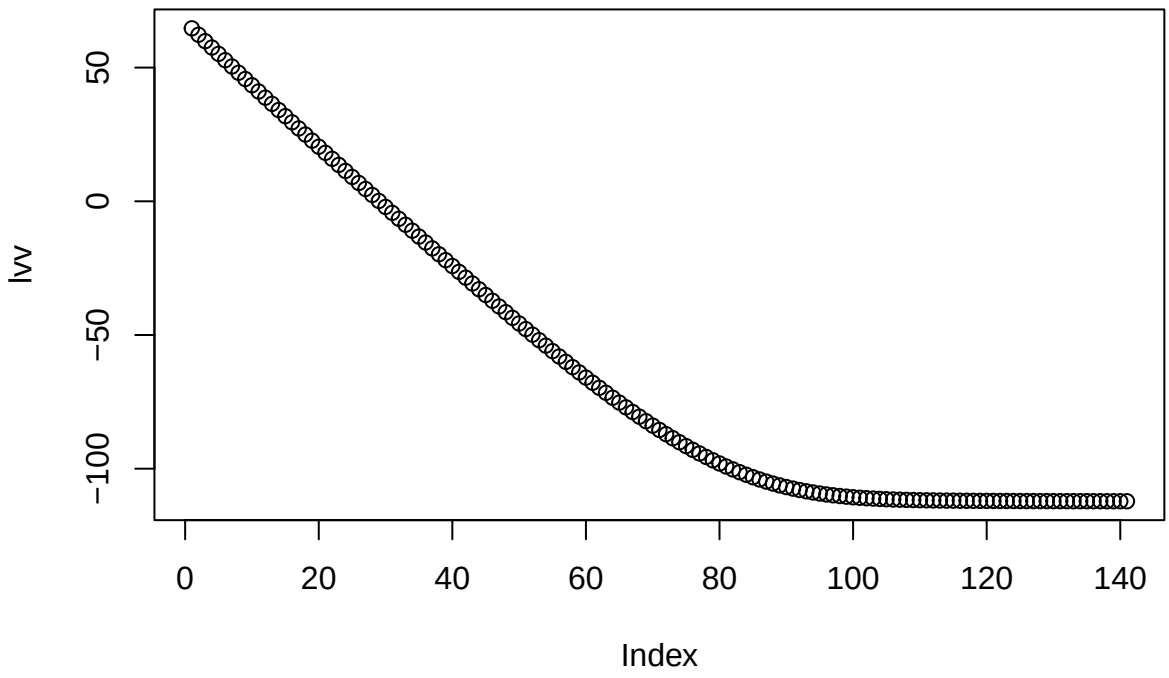

```
#Fix theta2 = biter[2]
out<-contour.plot.GP(80,-1.25,2,-10,1,2,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=40,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[3]),col="blue",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
bM4<-bM2[,c(1,3)]
st<-stop.num(bM4,.0001)
for(i in 2:(min(st))) {
   arrows(bM4[i-1,1],bM4[i-1,2],bM4[i,1],bM4[i,2],length=.05,col="purple")}
```




```
#####Theta1=biter[1]
out<-contour.plot.GP(80,-1.6,0.5,-10,1,1,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=40,drawlabels=TRUE,
     xlab=expression(theta[2]),ylab=expression(theta[3]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
bM5<-bM2[,c(2,3)]
st<-stop.num(bM5,.0001)
for(i in 2:(min(st))) {
   arrows(bM5[i-1,1],bM5[i-1,2],bM5[i,1],bM5[i,2],length=.05,col="purple")}
```


70
60
50
40
30
20
10
0
−10
−20
−30
−40
−50
−60
−70
−80
−90
−100
−110
−90
0
−2
−4
−6
−8
−10
σ3
−1.5
−1.0
−0.5
0.0
0.5
θ2

```
plot(lvv)
```

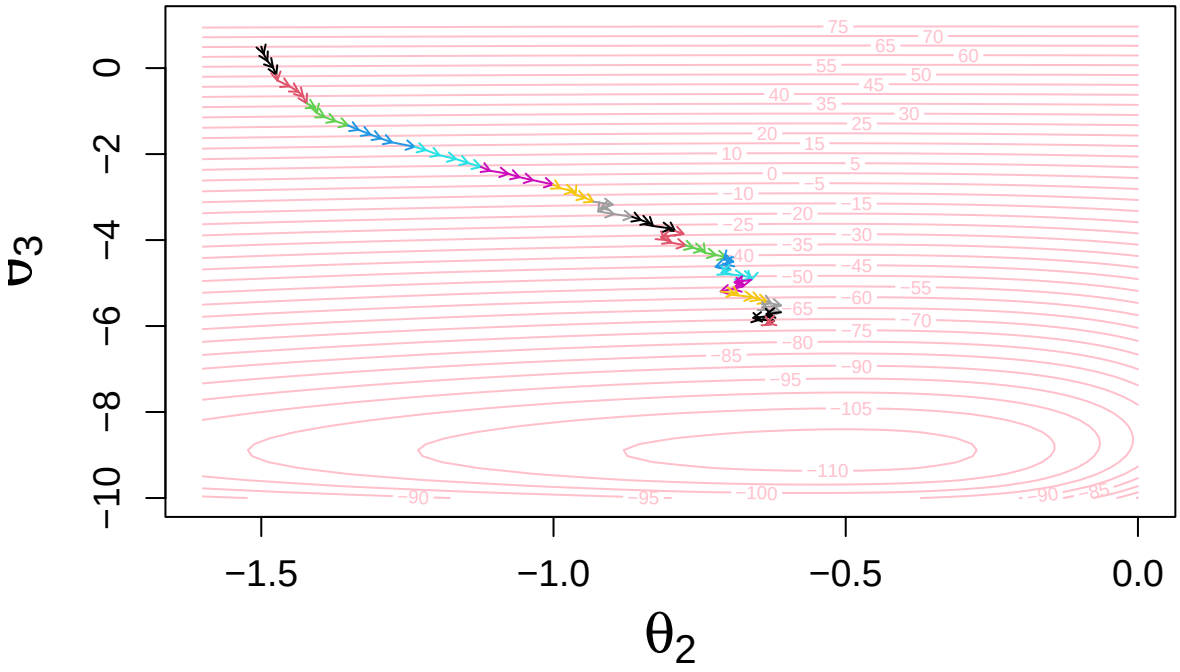


Gradient Descent on all three hyper-parameters leads to an estimate which maximizes the log-likelihood.

Stochastic: Incorporating stochastic gradient descent with 5 batches (of size 10). In the contour plot, each epoch is coded a different color. We also made the learning rate decrease, because a fixed learning rate kept bouncing around instead of converging.

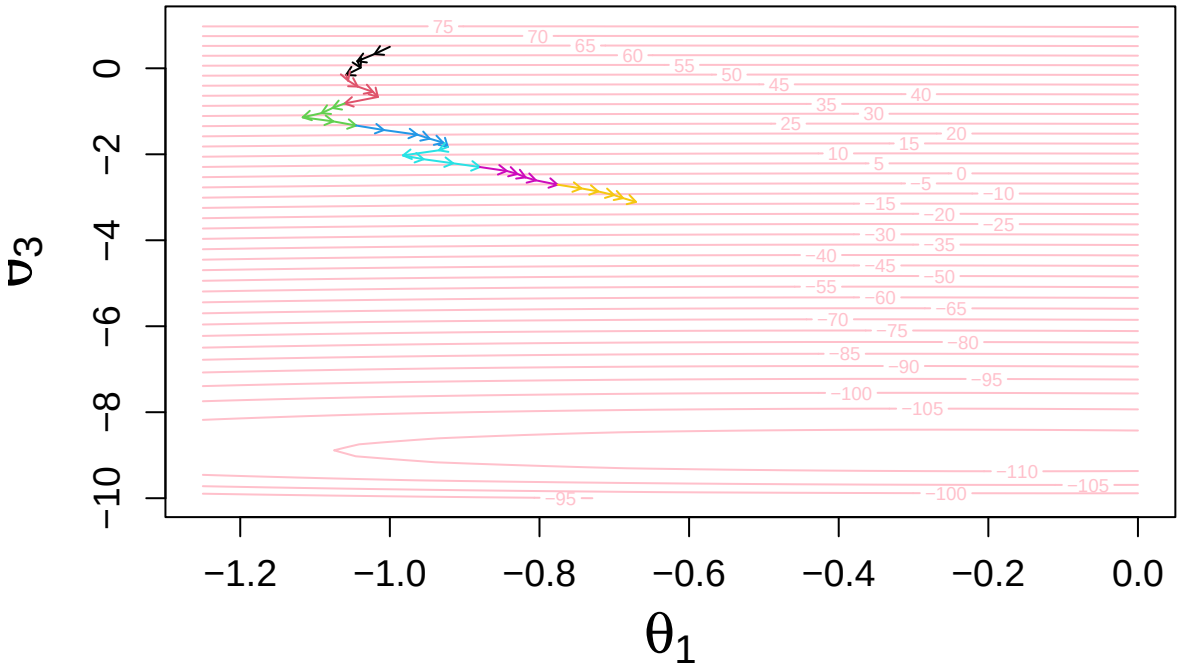
```
set.seed(1106)
n.s<-1
niter<-40
num.batch<-5
len<-length(Y)
batch.size<-len/num.batch
color.epoch<-c(rep(c(1:niter),each=num.batch))
lrat = rep(NA,niter*num.batch)#learning rate
lrat[1]<-0.05
for (t in 2:(niter*batch.size)){
lrat[t] = lrat[1]/(1+0.01*t) #learning rate
}
result<-theta.grad(x,Y,theta1.0,theta2.0,theta3.0,niter,
                   lrat,stochastic=1,num.batch,momentum=NULL,gamma=NULL)
bM2<-result[[1]]
gr<-result[[2]]
lvv<-result[[3]]
biter<-bM2[num.batch*niter+1,]
par(mfrow=c(1,1))
#####Fix Theta 1 = biter[1] and look at the contour
out<-contour.plot.GP(80,-1.6,0.0,-10,1,1,biter,x,Y)
```

```
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=40,drawlabels=TRUE,
     xlab=expression(theta[2]),ylab=expression(theta[3]),col="pink",
           cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM2[,(3*(o-1)+2):(3*(o-1)+3)]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=color.epoch[i])}
}
```

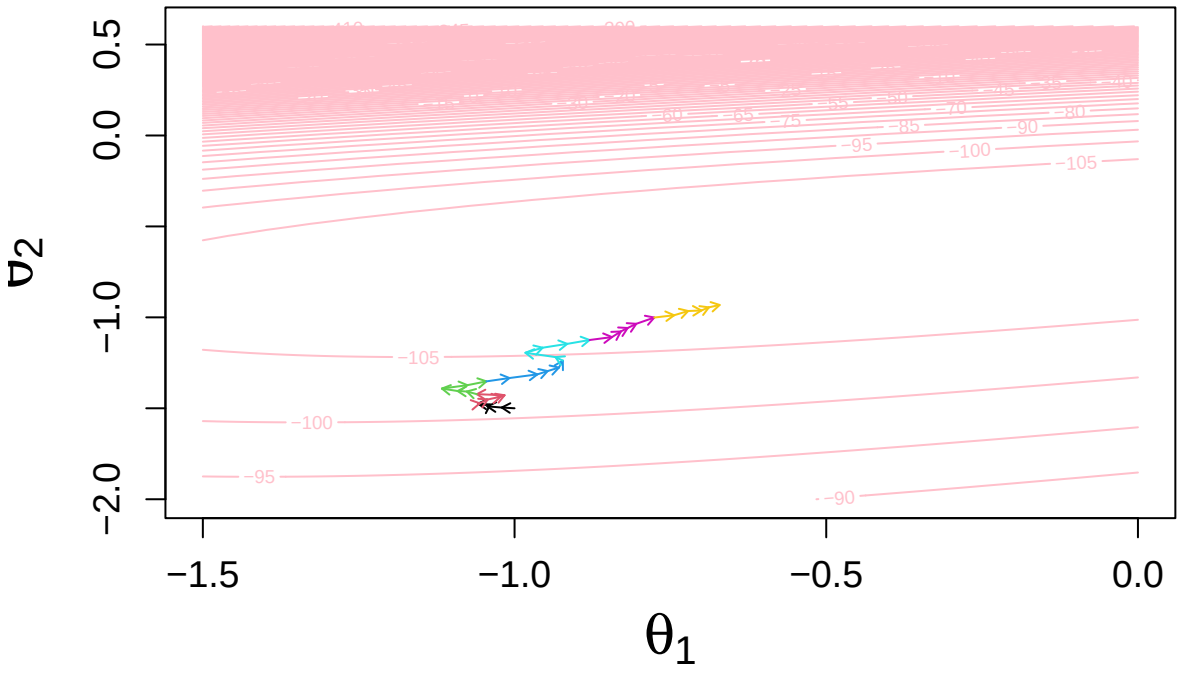



```
#####Fix Theta 2 = biter[2] and look at the contour
out<-contour.plot.GP(80,-1.25,0.0,-10,1,2,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=40,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[3]),col="pink",
           cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM2[,c((3*(o-1)+1),(3*(o-1)+3))]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
```

```
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=color.epoch[i])}
}
```




```
#####Fix Theta 3 = biter[3] and look at the contour
out<-contour.plot.GP(80,-1.5,0,-2,0.6,3,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=100,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[2]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM2[,c((3*(o-1)+1),(3*(o-1)+2))]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=color.epoch[i])}
}
```

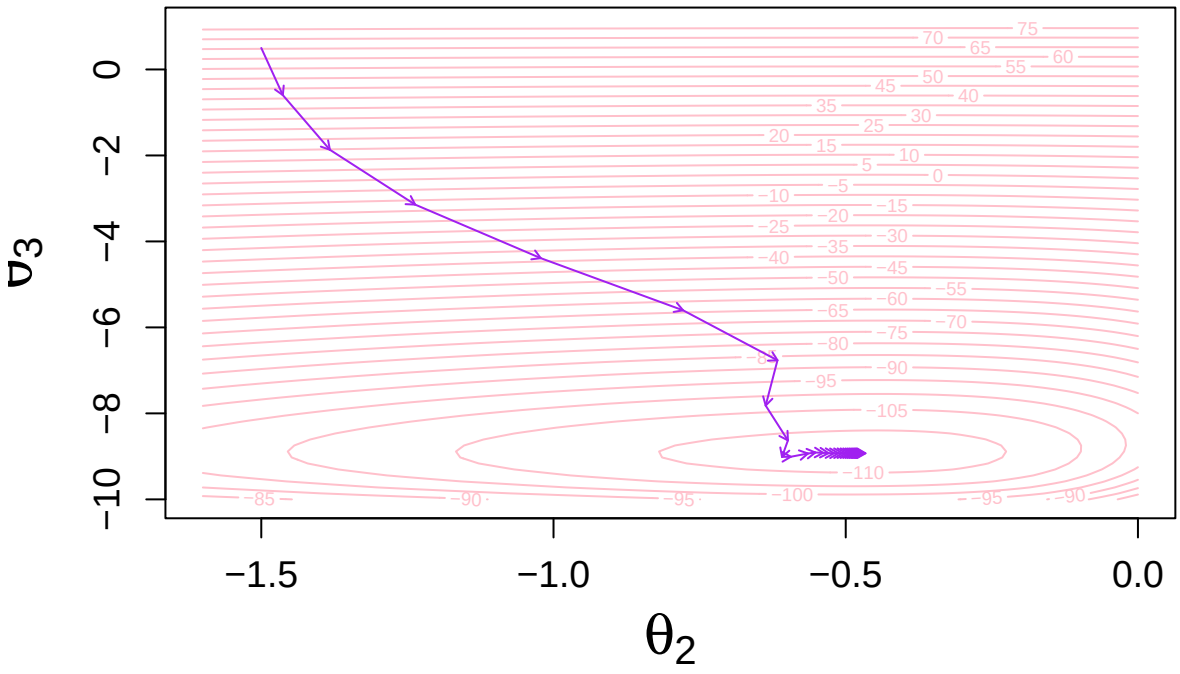


Looking at the contour plots, it looks like the code works since they converge in the trough.

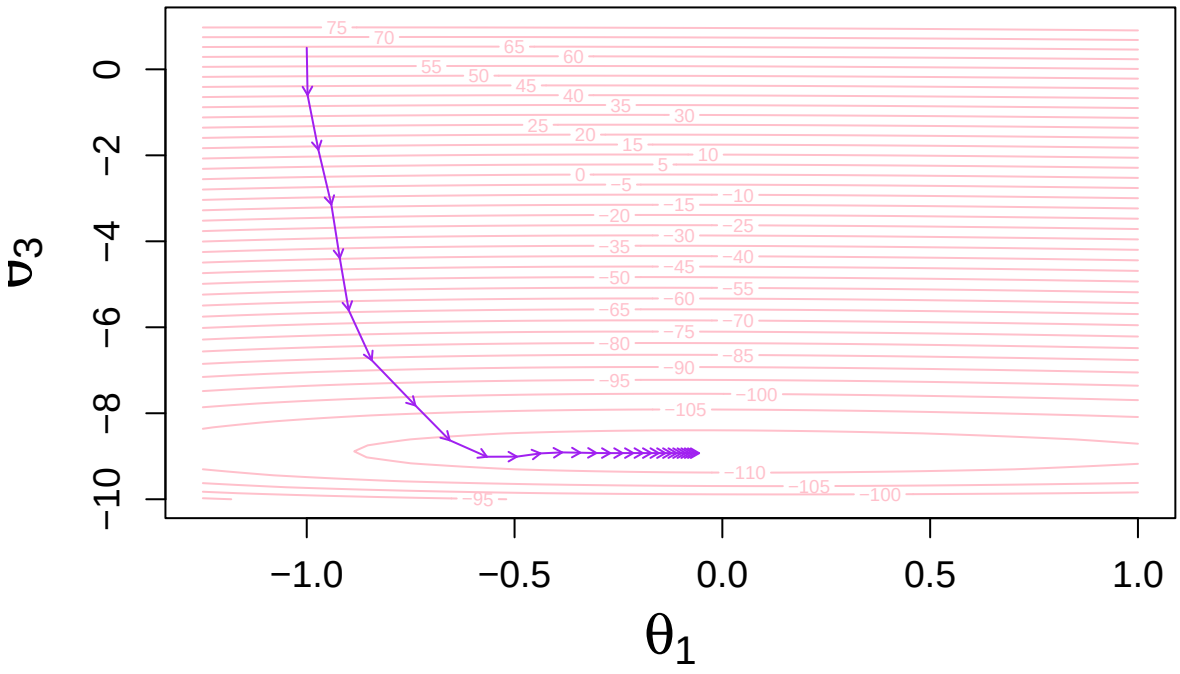
#Momentum: Here we did gradient descent with momentum, with gamma = 0.2

```
set.seed(1106)
n.s<-1
biter<-c(theta1.0,theta2.0,theta3.0)
niter<-40
num.batch<-1
batch.size<-len/num.batch
#color.epoch<-c(rep(c(1:niter),each=num.batch))
lrat.u = rep(0.05,niter*num.batch)#learning rate
for (t in 2:(niter*batch.size)){
lrat.u[t] = lrat.u[1]/(1+0.01*t) #learning rate
}
result<-theta.grad(x,Y,theta1.0,theta2.0,theta3.0,niter,lrat.u,
                   stochastic=0,num.batch=NULL,momentum=1,gamma=0.2)
bM2<-result[[1]]
gr<-result[[2]]
lvv<-result[[3]]
biter<-bM2[num.batch*niter+1,]
par(mfrow=c(1,1))
#####Fix Theta 1 = biter[1] and look at the contour
out<-contour.plot.GP(80,-1.6,0.0,-10,1,1,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
```

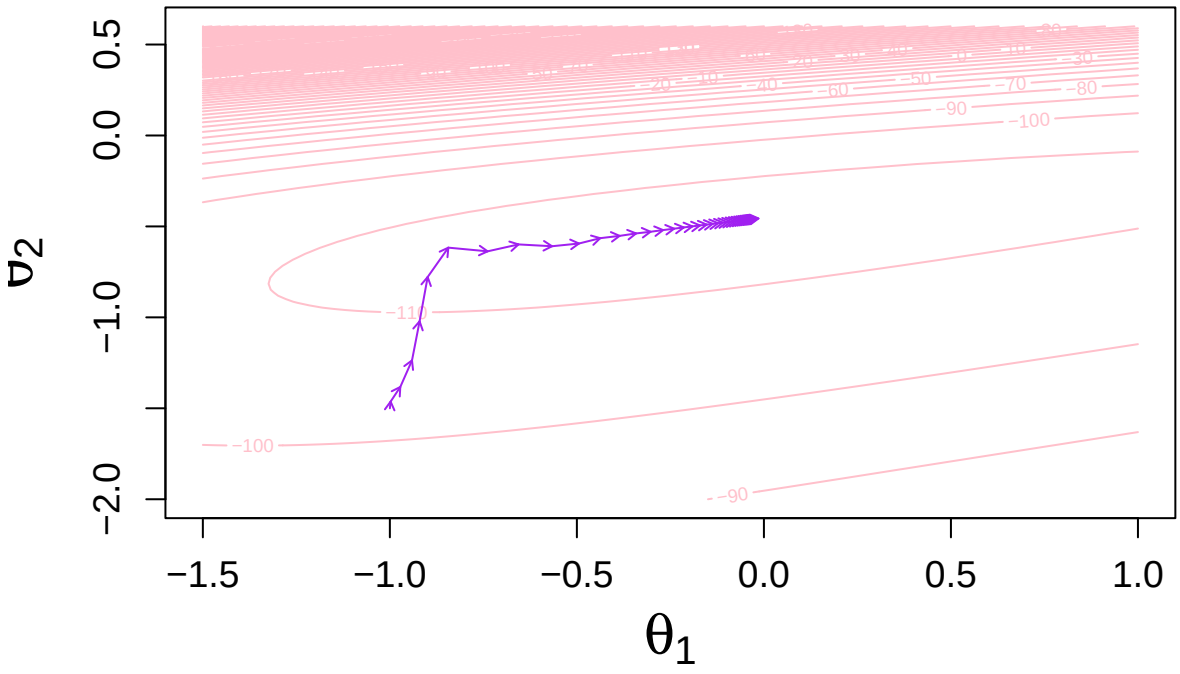
```
contour(alga1,alga2,llmat,nlevels=40,drawlabels=TRUE,
     xlab=expression(theta[2]),ylab=expression(theta[3]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM2[,(3*(o-1)+2):(3*(o-1)+3)]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col="purple")}
}
```




```
#####Fix Theta 2 = biter[2] and look at the contour
out<-contour.plot.GP(80,-1.25,1,-10,1,2,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=40,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[3]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM2[,c((3*(o-1)+1),(3*(o-1)+3))]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col="purple")}
}
```




```
#####Fix Theta 3 = biter[3] and look at the contour
out<-contour.plot.GP(80,-1.5,1,-2,0.6,3,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=100,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[2]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM2[,c((3*(o-1)+1),(3*(o-1)+2))]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col="purple")}
}
```

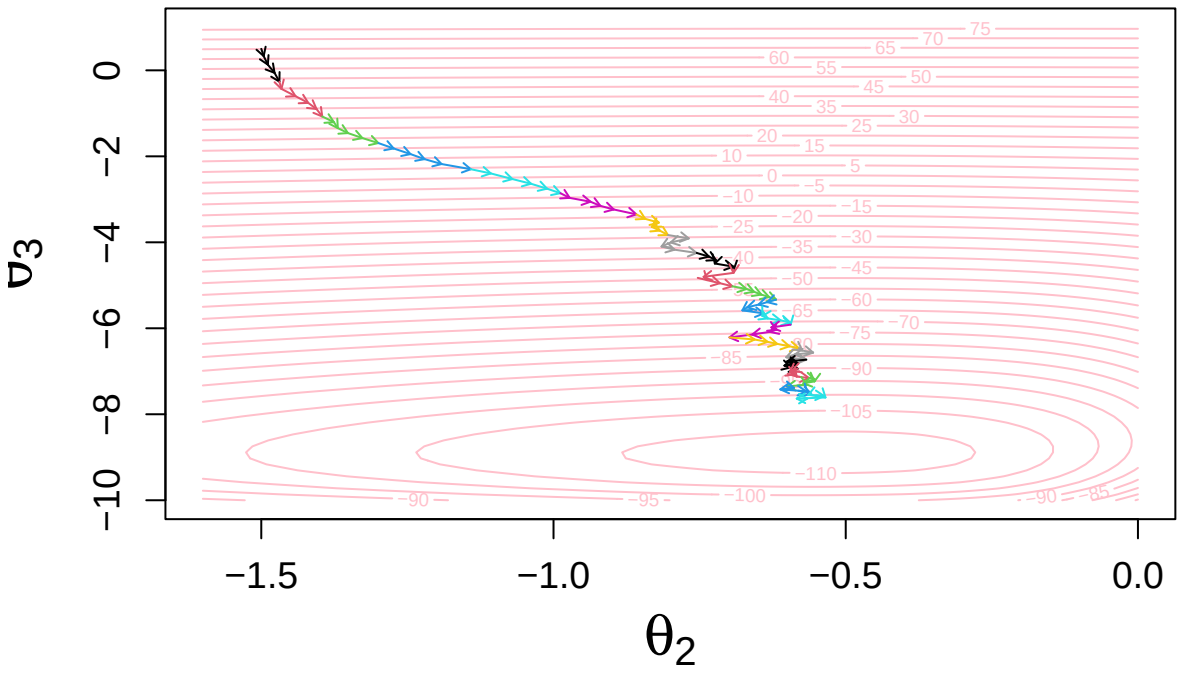


The code appears to work, since it converges in trough.

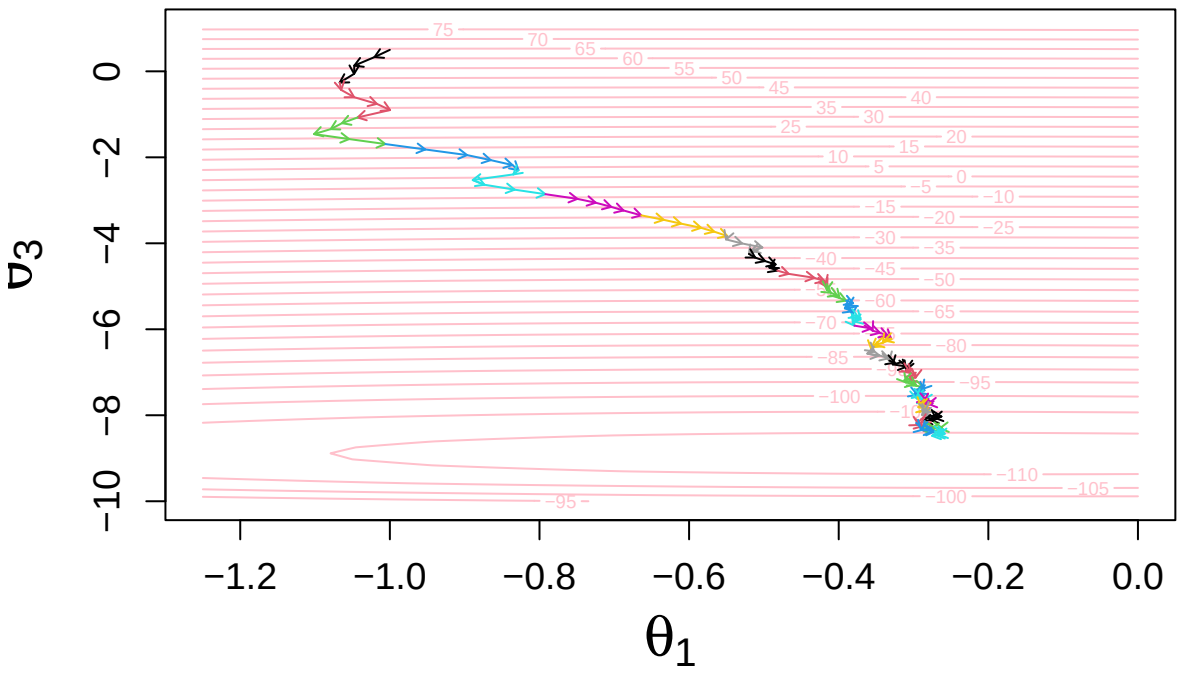
#With Momentum and Stochastic. Momentum with gamma equal to 0.2, and 5 batches per epoch (of size 10.)

```
set.seed(1106)
n.s<-1
biter<-c(theta1.0,theta2.0,theta3.0)
niter<-30
num.batch<-5
batch.size<-len/num.batch
color.epoch<-c(rep(c(1:niter),each=num.batch))
lrat.u = rep(0.05,niter*num.batch)#learning rate
for (t in 2:(niter*batch.size)){
lrat.u[t] = lrat.u[1]/(1+0.01*t) #learning rate
}
result<-theta.grad(x,Y,theta1.0,theta2.0,theta3.0,niter,lrat.u,
                   stochastic=1,num.batch=5,momentum=1,gamma=0.2)
bM2<-result[[1]]
gr<-result[[2]]
lvv<-result[[3]]
biter<-bM2[num.batch*niter+1,]
par(mfrow=c(1,1))
#####Fix Theta 1 = biter[1] and look at the contour
out<-contour.plot.GP(80,-1.6,0.0,-10,1,1,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
```

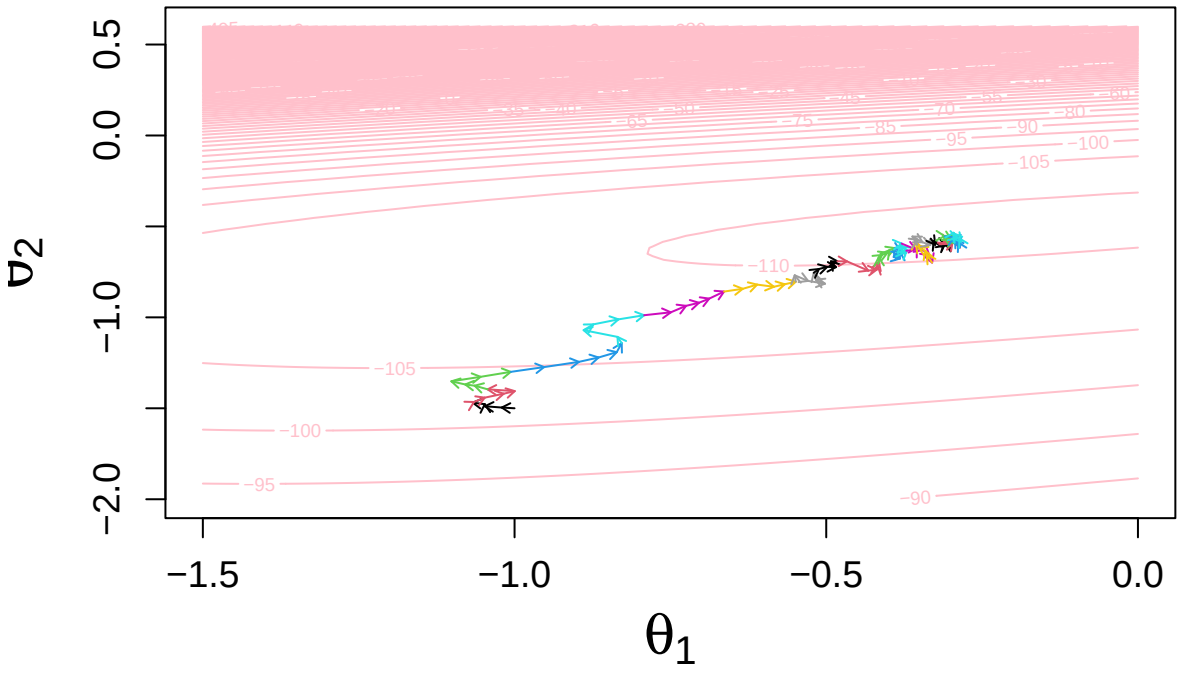
```
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=40,drawlabels=TRUE,
     xlab=expression(theta[2]),ylab=expression(theta[3]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM2[,(3*(o-1)+2):(3*(o-1)+3)]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=color.epoch[i])}
}
```




```
#####Fix Theta 2 = biter[2] and look at the contour
out<-contour.plot.GP(80,-1.25,0.0,-10,1,2,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=40,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[3]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM2[,c((3*(o-1)+1),(3*(o-1)+3))]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=color.epoch[i])}
}
```




```
#####Fix Theta 3 = biter[3] and look at the contour
out<-contour.plot.GP(80,-1.5,0,-2,0.6,3,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=100,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[2]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM2[,c((3*(o-1)+1),(3*(o-1)+2))]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=color.epoch[i])}
}
```

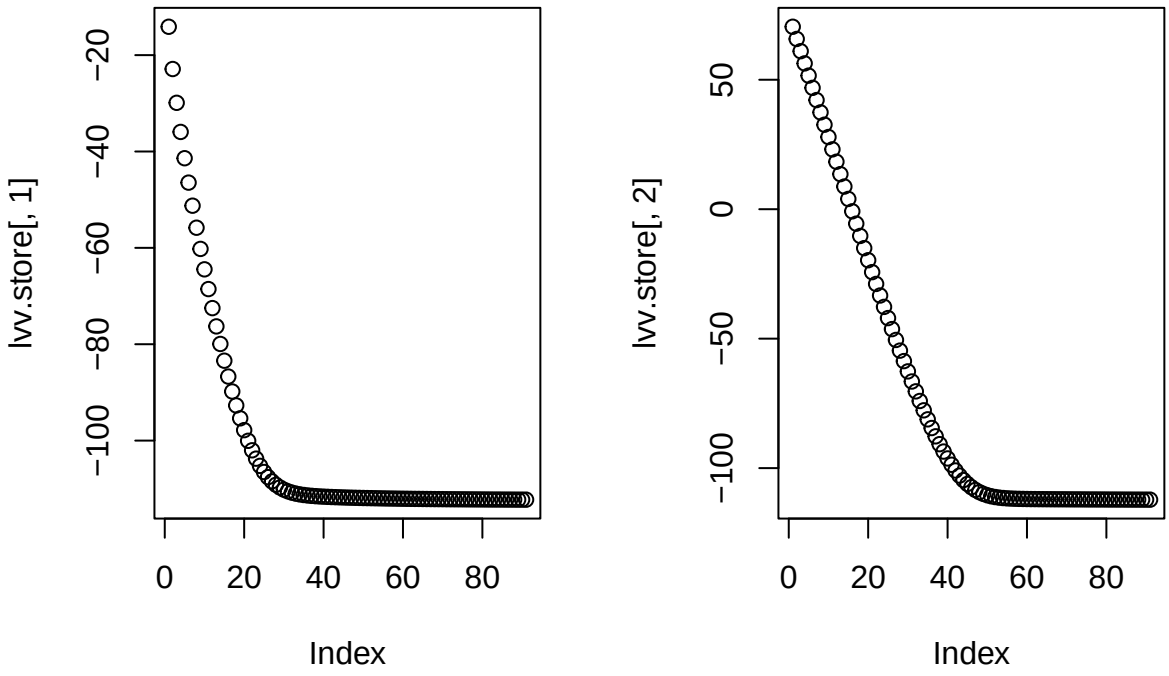


The code appears to work, since they converge to the trough.

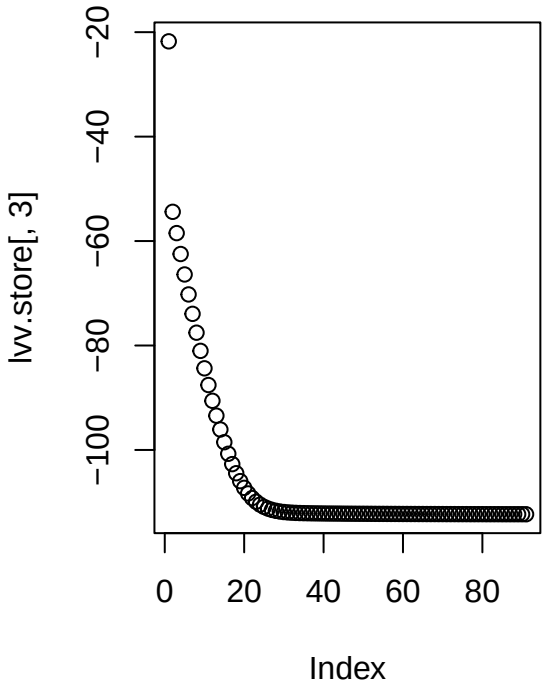
#Comparing with different start values. In the following graphs, gradient descent was conducted for various different initial values for the three parameters (theta 1, theta 2, theta 3). To visualize the results, contour plots were created using the mean of the estimates from each set of initial theta values for the fixed value in each contour plot.

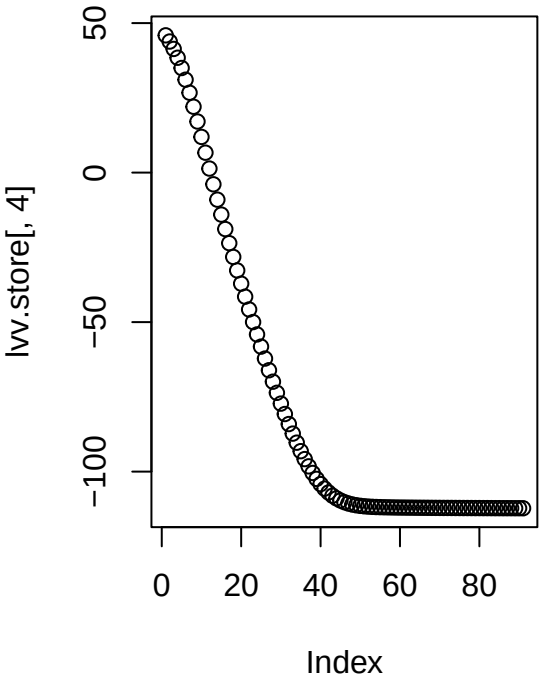
```
n.s=6
iterations<-c(90,90,90,90,90,90)
iter.max<-max(iterations)
biter.store<-matrix(rep(NA),nrow=n.s,ncol=3)
lvv.store<-matrix(rep(NA),nrow=iter.max+1,ncol=n.s)
gr.store<-matrix(rep(NA),nrow=n.s,ncol=3)
bM.store<-matrix(rep(NA),nrow=iter.max+1,ncol=(3*n.s))
theta1.0<-c(-2.8,2,-1,-2.8,1.75,0.5)
theta2.0<-c(-1.5,-1.25,0.2,0.4,0.2,-2.5)
theta3.0<-c(-4,0.5,-5,-1,-9.5,0.1)
lrat=c(rep(0.01,iter.max))
for (o in 1:n.s){
result<-theta.grad(x,Y,theta1.0[o],theta2.0[o],theta3.0[o],iterations[o],lrat)
bM.store[1:(iterations[o]+1),(3*(o-1)+1):(3*(o-1)+3)]<-as.matrix(result[[1]])
gr.store[o,]<-result[[2]]
lvv.store[1:(iterations[o]+1),o]<-result[[3]]
biter.store[o,]<-bM.store[iterations[o]+1,(3*(o-1)+1):(3*(o-1)+3)]
}

colors<-c("purple","blue","red","green","orange","brown")
```

```
biter<-c(mean(biter.store[,1]),mean(biter.store[,2]),mean(biter.store[,3]))
par(mfrow=c(1,2))
plot (lvv.store[,1])
plot (lvv.store[,2])
```

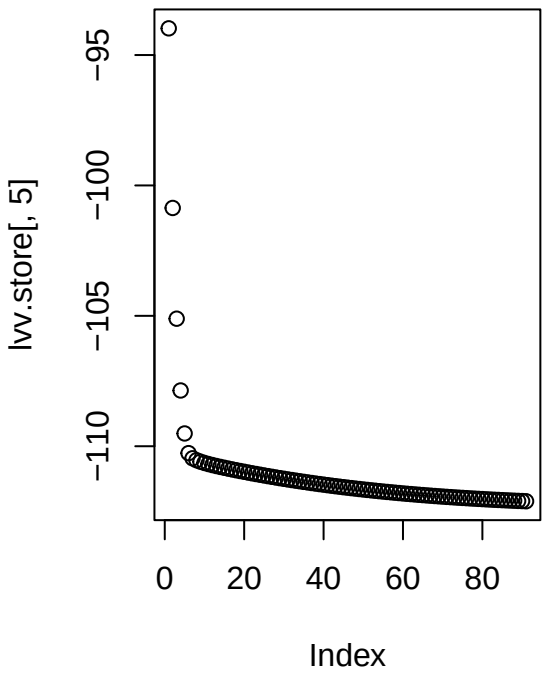


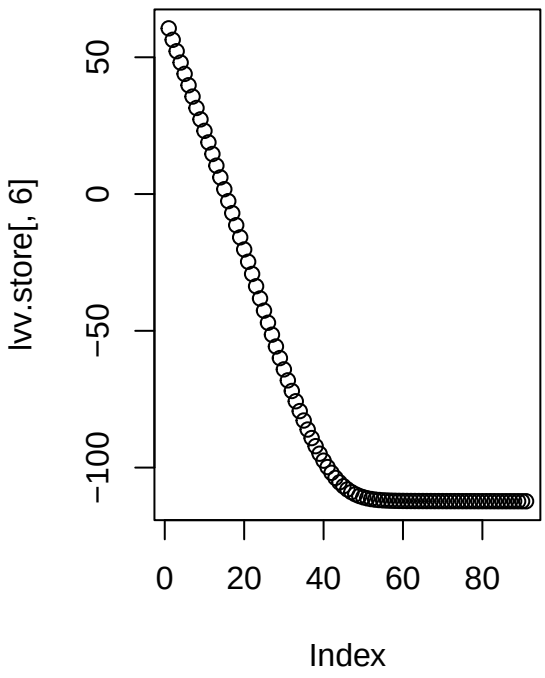

```
plot (lvv.store[,3])
plot (lvv.store[,4])
```



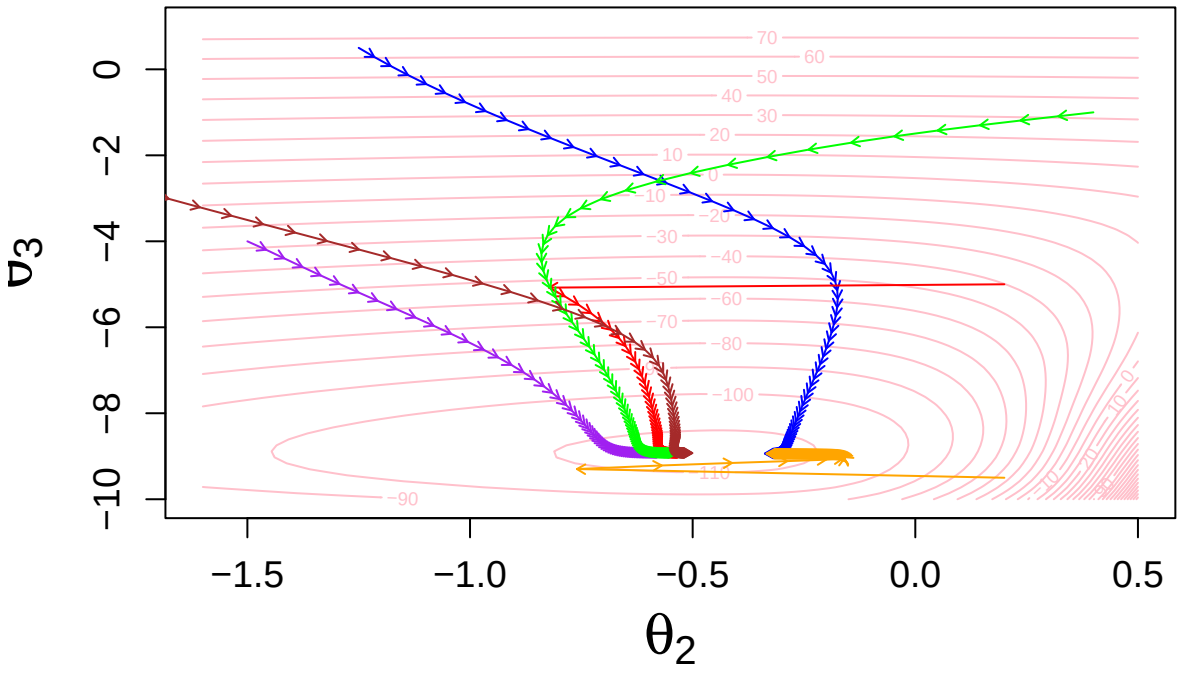



```
plot (lvv.store[,5])
plot (lvv.store[,6])
```

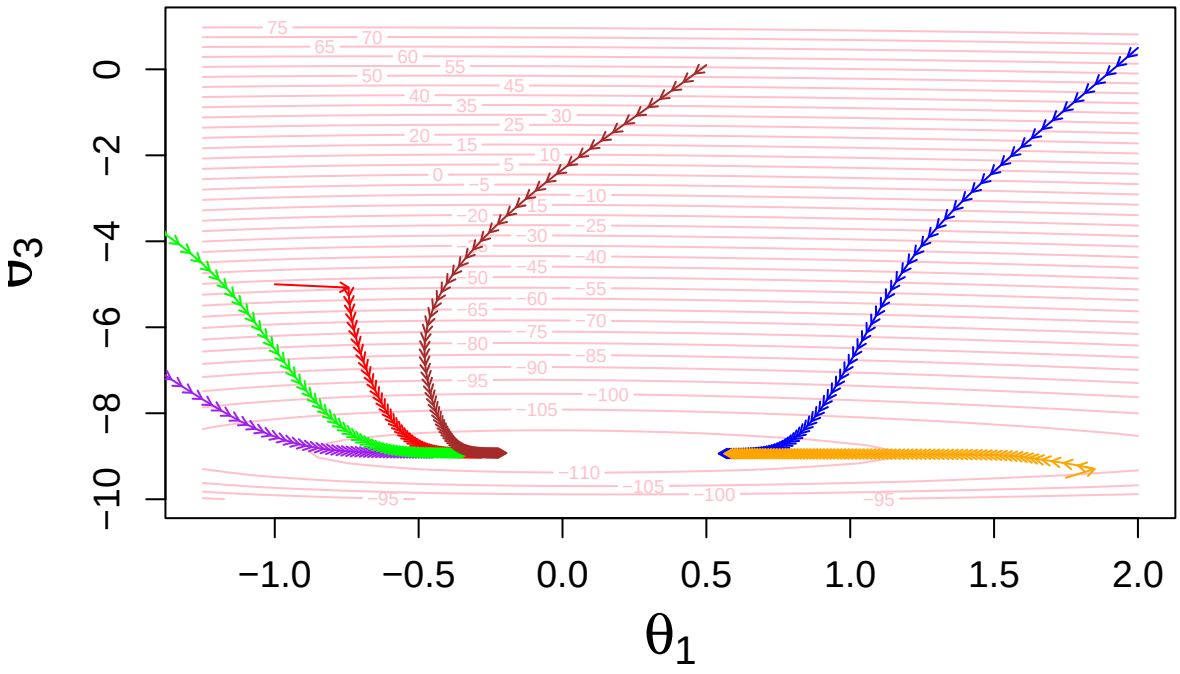




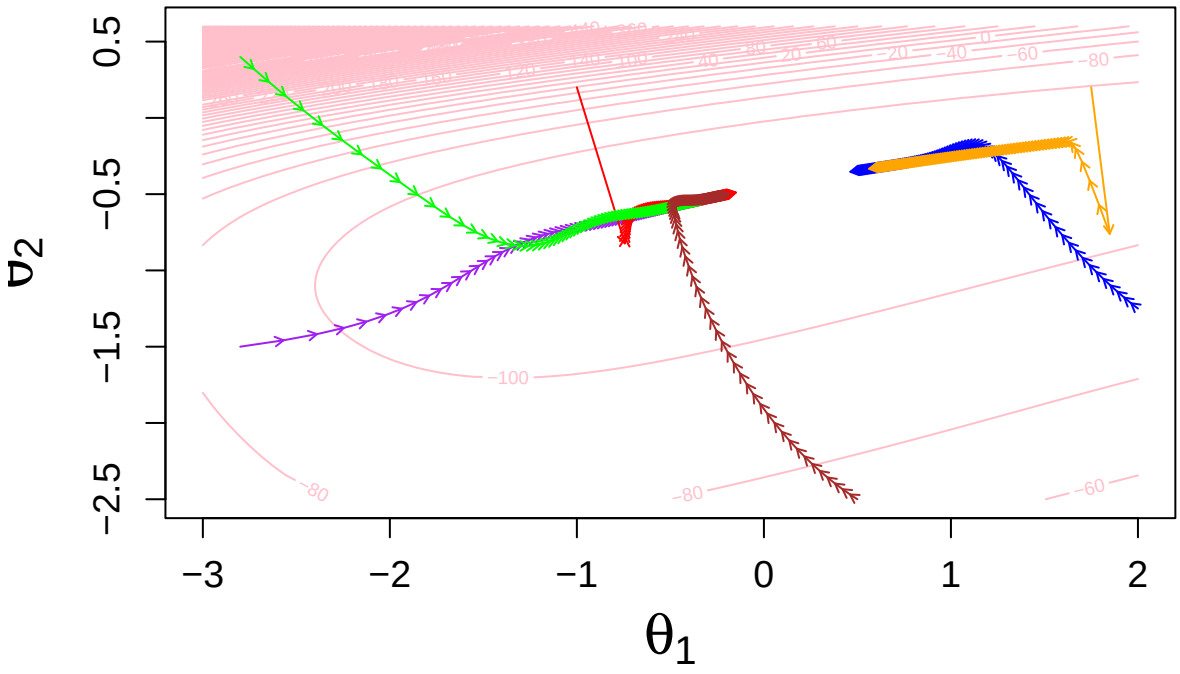

```
par(mfrow=c(1,1))
#####Fix Theta 1 = biter[1] and look at the contour
out<-contour.plot.GP(80,-1.6,0.5,-10,1,1,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=40,drawlabels=TRUE,
     xlab=expression(theta[2]),ylab=expression(theta[3]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM.store[(1:iterations[o]),(3*(o-1)+2):(3*(o-1)+3)]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=colors[o])}
}
```




```
#####Fix Theta 2 = biter[2] and look at the contour
out<-contour.plot.GP(80,-1.25,2,-10,1,2,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=40,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[3]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM.store[(1:iterations[o]),c((3*(o-1)+1),(3*(o-1)+3))]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=colors[o])}
}
```




```
#####Fix Theta 3 = biter[3] and look at the contour
out<-contour.plot.GP(80,-3,2,-2.5,0.6,3,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=100,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[2]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM.store[(1:iterations[o]),c((3*(o-1)+1),(3*(o-1)+2))]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=colors[o])}
}
```




```
gr.store
```

```
##             [,1]        [,2]         [,3]
## [1,]  0.5360926  0.14236936 -0.006107464
## [2,] -0.6674304 -0.13224050  0.010729617
## [3,]  0.2897403  0.07264482 -0.004075590
## [4,]  0.5746208  0.15385186 -0.007317213
## [5,] -0.7973458 -0.15301865  0.014921584
## [6,]  0.3528386  0.08969192 -0.006823873
```

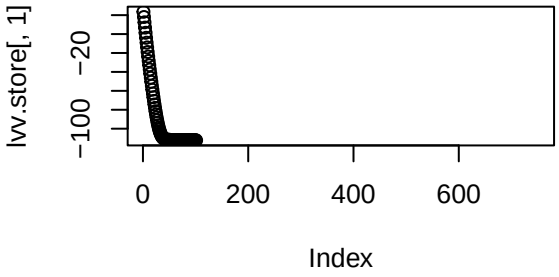
```
biter.store
```

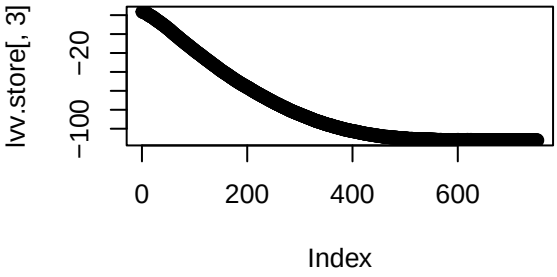
```
##             [,1]       [,2]      [,3]
## [1,] -0.2688352 -0.5199781 -8.925457
## [2,]  0.4488164 -0.3561592 -8.937285
## [3,] -0.1438235 -0.4877473 -8.927055
## [4,] -0.2875443 -0.5249591 -8.925204
## [5,]  0.5463909 -0.3371168 -8.939194
## [6,] -0.1767471 -0.4960537 -8.926514
```

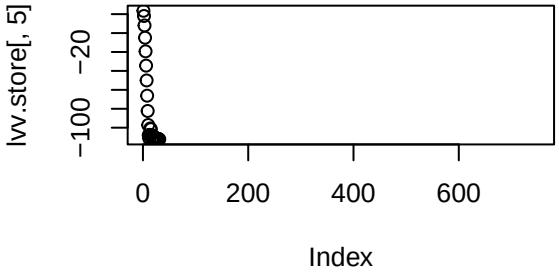
From all the starting points, gradient descent appears to be converging to approximetly the same estimate.

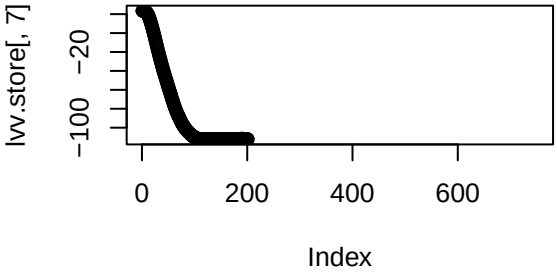
#Comparing the methods.

```
n.s=8
#First 2 regular (green)
#Next 2 Stochastic (blue)
#Next 2 Momentum (orange)
#Last 2 Momentum and Stochastic (Red)
iterations<-c(100,100,150,150,30,30,40,60)
num.batch.setup<-c(1,1,5,5,1,1,5,5)
iter.max<-max(iterations*num.batch.setup)
biter.store<-matrix(rep(NA),nrow=n.s,ncol=3)
lvv.store<-matrix(rep(NA),nrow=iter.max+1,ncol=n.s)
gr.store<-matrix(rep(NA),nrow=n.s,ncol=3)
bM.store<-matrix(rep(NA),nrow=iter.max+1,ncol=(3*n.s))
theta1.0<-c(-2.8,1.75,-2.8,1.75,-2.8,1.75,-2.8,1.75)
theta2.0<-c(-1.5,0.4,-1.5,0.4,-1.5,0.4,-1.5,0.4)
theta3.0<-c(-2,0.1,-2,0.1,-2,0.1,-2,0.1)
mom.setup<-c(0,0,0,0,1,1,1,1)
st.setup<-c(0,0,1,1,0,0,1,1)
lrat=c(rep(0.01,iter.max))
for (o in 1:n.s){
set.seed(1234)
b<-st.setup[o]
c<-mom.setup[o]
result<-theta.grad(x,Y,theta1.0[o],theta2.0[o],theta3.0[o],iterations[o],
                   lrat,stochastic=b,num.batch=num.batch.setup[o],momentum=c,gamma=0.8)
bM.store[1:(iterations[o]*num.batch.setup[o]+1),(3*(o-1)+1):(3*(o-1)+3)]<-as.matrix(result[[1]])
gr.store[o,]<-result[[2]]
lvv.store[1:(iterations[o]*num.batch.setup[o]+1),o]<-result[[3]]
biter.store[o,]<-bM.store[iterations[o]*num.batch.setup[o]+1,(3*(o-1)+1):(3*(o-1)+3)]
}

colors<-c("green","green","blue","blue","orange","orange","red","red")
biter<-c(mean(biter.store[,1]),mean(biter.store[,2]),mean(biter.store[,3]))
par(mfrow=c(2,2))
plot (lvv.store[,1])
plot (lvv.store[,3])
plot (lvv.store[,5])
plot (lvv.store[,7])
```

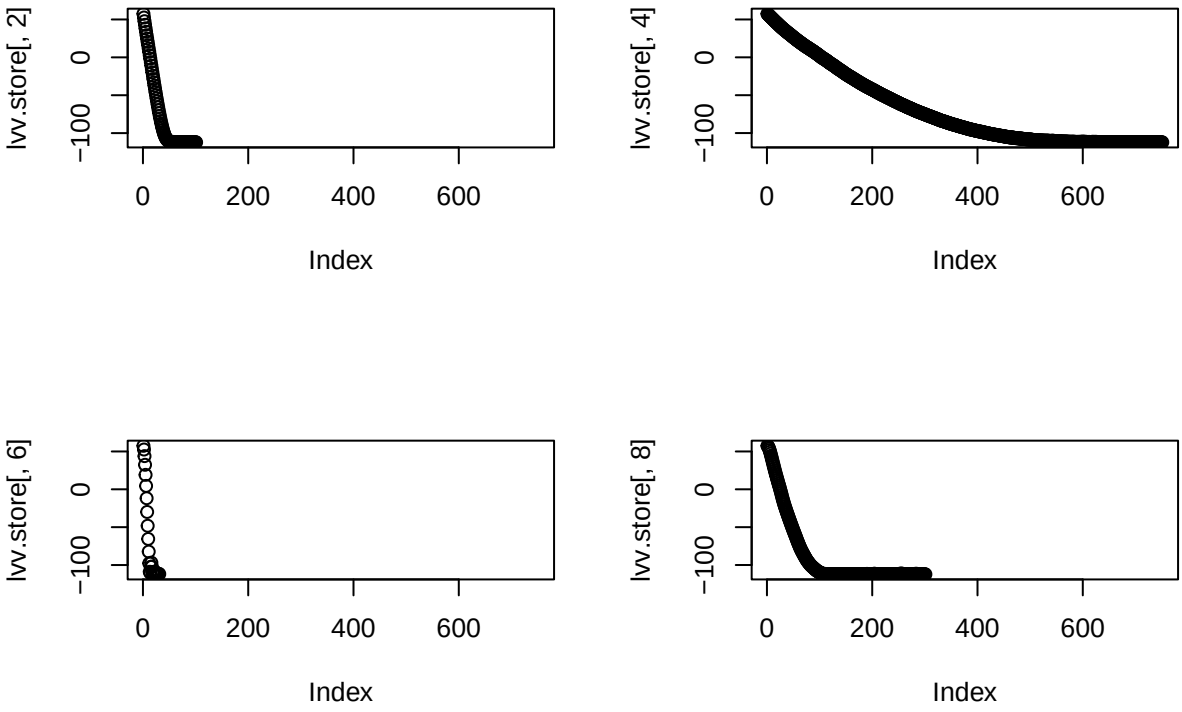




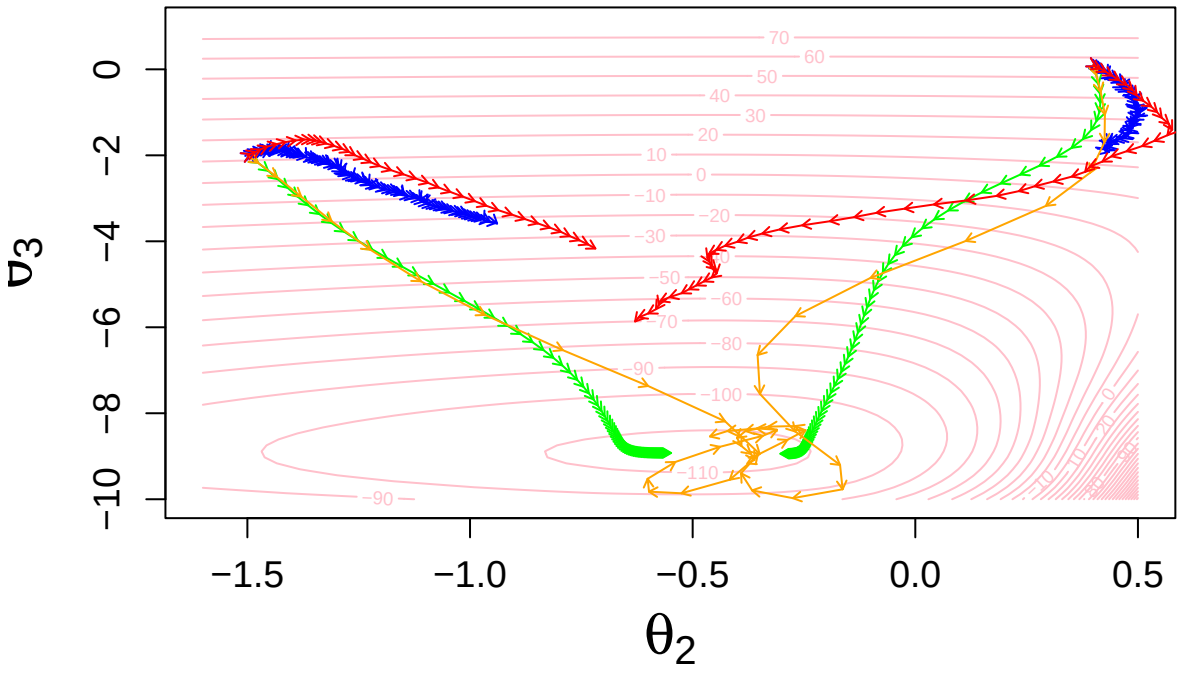




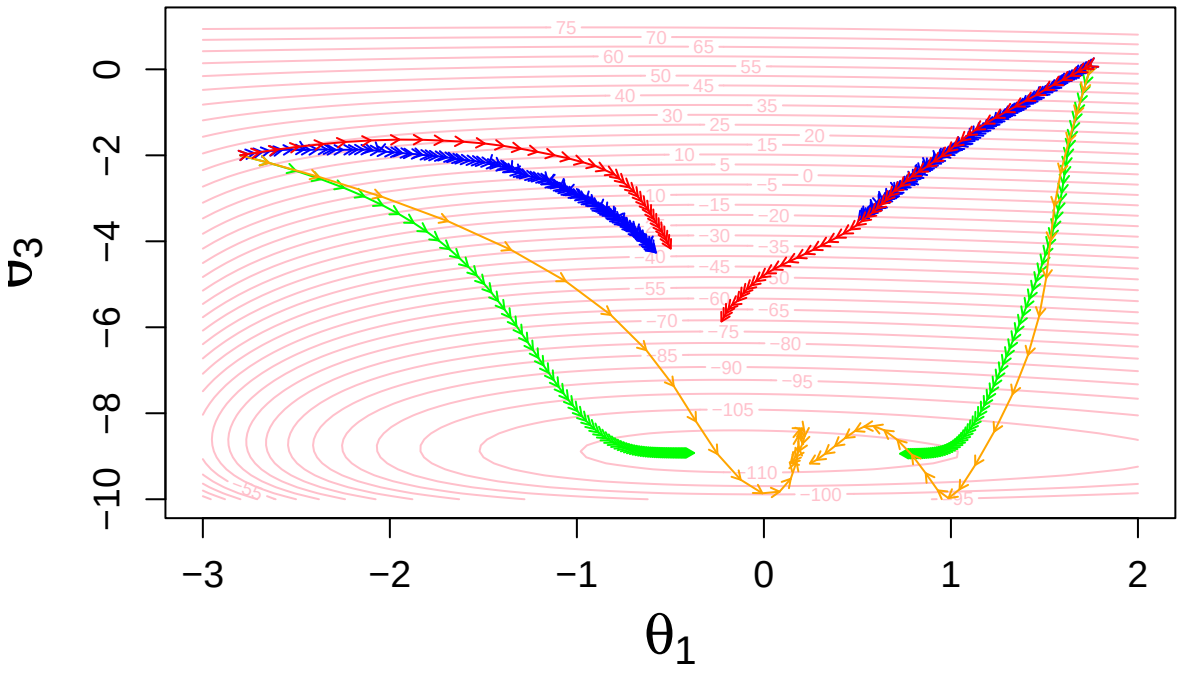

```
plot (lvv.store[,2])
plot (lvv.store[,4])
plot (lvv.store[,6])
plot (lvv.store[,8])
```



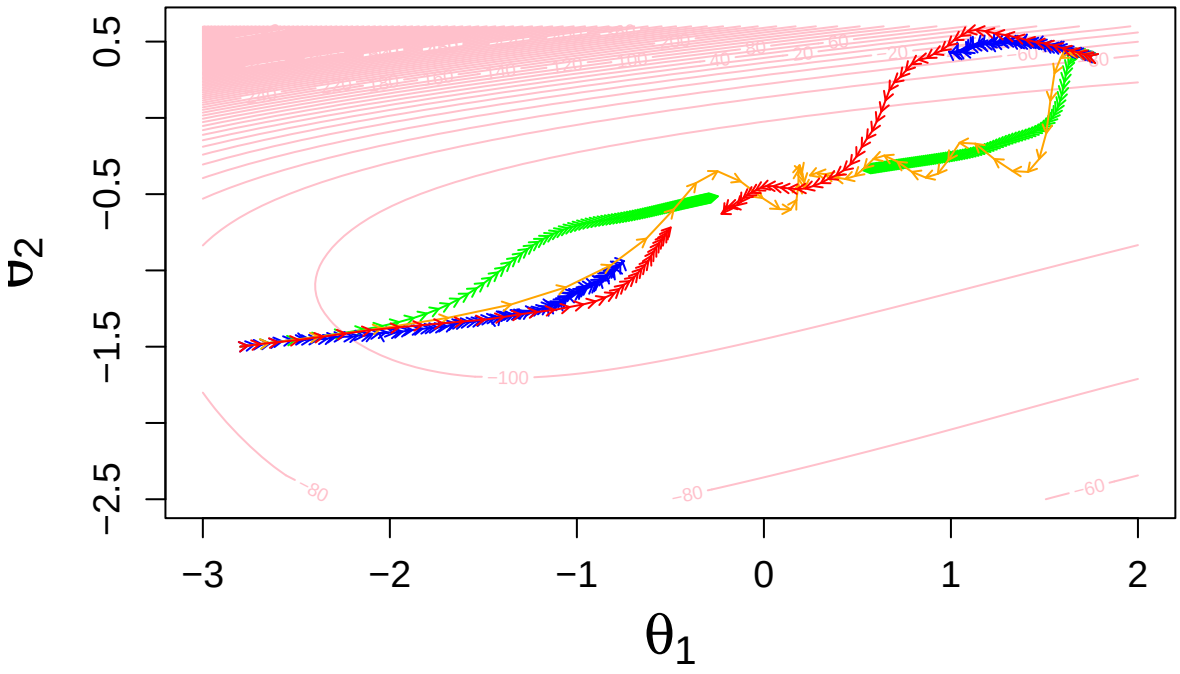

```
par(mfrow=c(1,1))
#####Fix Theta 1 = biter[1] and look at the contour
out<-contour.plot.GP(80,-1.6,0.5,-10,1,1,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=40,drawlabels=TRUE,
     xlab=expression(theta[2]),ylab=expression(theta[3]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM.store[(1:iterations[o]),(3*(o-1)+2):(3*(o-1)+3)]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=colors[o])}
}
```




```
#####Fix Theta 2 = biter[2] and look at the contour
out<-contour.plot.GP(80,-3,2,-10,1,2,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=40,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[3]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM.store[(1:iterations[o]),c((3*(o-1)+1),(3*(o-1)+3))]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=colors[o])}
}
```




```
#####Fix Theta 3 = biter[3] and look at the contour
out<-contour.plot.GP(80,-3,2,-2.5,0.6,3,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=100,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[2]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM.store[(1:iterations[o]),c((3*(o-1)+1),(3*(o-1)+2))]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=colors[o])}
}
```




```
gr.store
```

```
##             [,1]       [,2]         [,3]
## [1,]  0.47053328  0.1231080 -0.005714614
## [2,] -0.74879697 -0.1454085  0.013916684
## [3,]  0.66544704 -3.4748871 -0.567691747
## [4,]  0.66141034 -3.4615258 -0.574516906
## [5,] -0.46223089  1.3420425  3.075747711
## [6,] -0.21619671 -0.2029454  5.465928328
## [7,]  1.10976864 -7.7486083  2.899105328
## [8,]  0.07246942  0.3180422 -0.742698104
```

```
biter.store
```

```
##            [,1]       [,2]      [,3]
## [1,] -0.2364827 -0.5114598 -8.925838
## [2,]  0.5093289 -0.3442799 -8.938493
## [3,] -0.2614596 -0.5570606 -8.888734
## [4,] -0.2609022 -0.5564635 -8.867661
## [5,]  0.1245886 -0.4589011 -9.003085
## [6,]  0.1930879 -0.4340594 -9.141049
## [7,] -0.2849267 -0.5729864 -8.900218
## [8,] -0.2602919 -0.5322389 -8.935524
```

```
sqrt(exp(biter.store))
```

```
##           [,1]      [,2]       [,3]
## [1,] 0.8884816 0.7743511 0.01152866
## [2,] 1.2900287 0.8418613 0.01145594
## [3,] 0.8774548 0.7568954 0.01174454
## [4,] 0.8776994 0.7571213 0.01186894
## [5,] 1.0642755 0.7949703 0.01109188
## [6,] 1.1013580 0.8049061 0.01035253
## [7,] 0.8672193 0.7508922 0.01167730
## [8,] 0.8779673 0.7663476 0.01147297
```

```
cbind(t1,mean(((biter.store[,1]))))
```

```
##             t1
## [1,] -1.386294 -0.05963221
```

```
cbind(t2,mean(((biter.store[,2]))))
```

```
##              t2
## [1,] -0.7133499 -0.4959312
```

```
cbind(t3,mean(((biter.store[,3]))))
```

```
##            t3
## [1,] -9.21034 -8.950075
```

All 4 methods appear to be converging to the same parameter values, but based on the gradients both the first and second parameters need to be run longer. We did this later (without the contour plots) because the contour plots couldn't handle the small arrows.

#####To compare how well they do.

```
n.s<-8
iterations<-c(500,500,100,100,500,500,100,100)
num.batch.setup<-c(1,1,5,5,1,1,5,5)
iter.max<-max(iterations*num.batch.setup)
max.loops<-100
loops.store<-c(NA,n.s)
biter.store<-matrix(rep(NA),nrow=n.s,ncol=3)
lvv.store<-matrix(rep(NA),nrow=iter.max+1,ncol=n.s)
gr.store<-matrix(rep(NA),nrow=n.s,ncol=3)
bM.store<-matrix(rep(NA),nrow=iter.max+1,ncol=(3*n.s))
theta1.0<-c(-2.8,1.75,-2.8,1.75,-2.8,1.75,-2.8,1.75)
theta2.0<-c(-1.5,0.4,-1.5,0.4,-1.5,0.4,-1.5,0.4)
theta3.0<-c(-2,0.1,-2,0.1,-2,0.1,-2,0.1)
mom.setup<-c(0,0,0,0,1,1,1,1)
st.setup<-c(0,0,1,1,0,0,1,1)
#lrat=c(rep(0.01,iter.max))
lrat.u = rep(0.01,iter.max*max(num.batch.setup))#learning rate
for (t in 2:(iter.max*max(num.batch.setup))){
```

```
lrat.u[t] = lrat.u[1]/(1+0.0001*t) #learning rate
}
for (o in 1:n.s){
set.seed(1234)
b<-st.setup[o]
c<-mom.setup[o]
i=1
gr.store[o,]<-c(1,1,1)
while((i<max.loops) &(sum(abs(gr.store[o,]))>0.5)){
result<-theta.grad(x,Y,theta1.0[o],theta2.0[o],theta3.0[o],iterations[o],lrat.u,
                   stochastic=b,num.batch=num.batch.setup[o],momentum=c,gamma=0.8)
bM.store[1:(iterations[o]*num.batch.setup[o]+1),(3*(o-1)+1):(3*(o-1)+3)]<-as.matrix(result[[1]])
gr.store[o,]<-result[[2]]
lvv.store[1:(iterations[o]*num.batch.setup[o]+1),o]<-result[[3]]
biter.store[o,]<-bM.store[iterations[o]*num.batch.setup[o]+1,(3*(o-1)+1):(3*(o-1)+3)]
theta1.0[o]<-biter.store[o,1]
theta2.0[o]<-biter.store[o,2]
theta3.0[o]<-biter.store[o,3]
i<-i+1
}
loops.store[o]<-i
}

true.theta<-c(t1,t2,t3)
rbind(true.theta,biter.store)
```

```
##                   [,1]       [,2]      [,3]
## true.theta -1.38629436 -0.7133499 -9.210340
##             0.01564143 -0.4491741 -8.929510
##             0.01666636 -0.4489348 -8.929527
##            -0.25161545 -0.5658347 -9.050753
##            -0.25161545 -0.5658347 -9.050753
##             0.01594271 -0.4491038 -8.929515
##             0.01594271 -0.4491038 -8.929515
##            -0.18589418 -0.6007290 -8.949236
##            -0.18589418 -0.6007290 -8.949236
```

```
loops.store
```

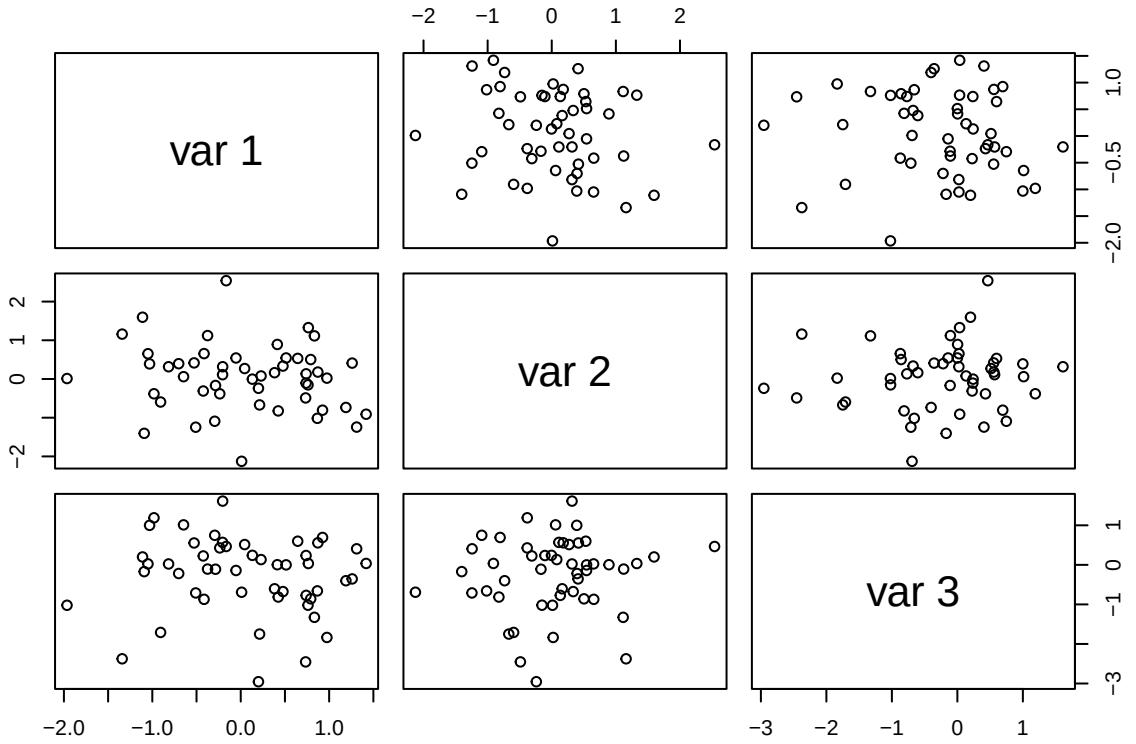
```
## [1]  2  2  9  9  2  2 12 12
```

The estimates (via all methods) are all reasonably close. When run until sum(|gradients|)<0.1, the estimates were very similar, but all four stochastic versions ran through 100 loops.

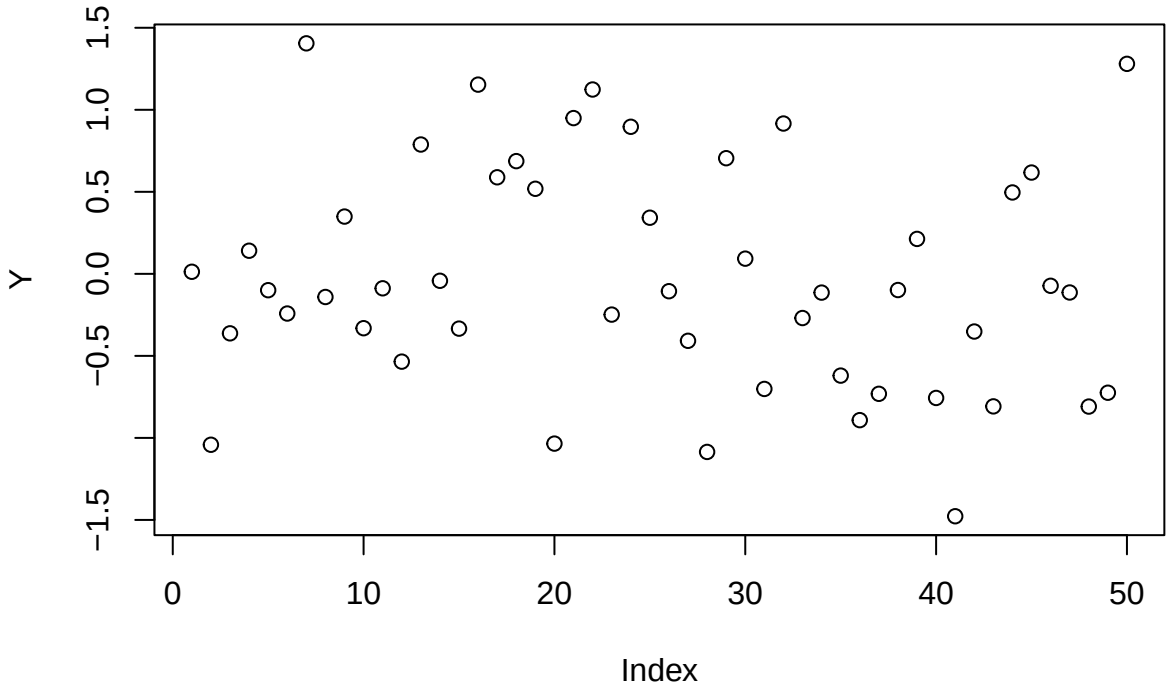
#Higher dimensional X

Regular gradient descent with different initial values.

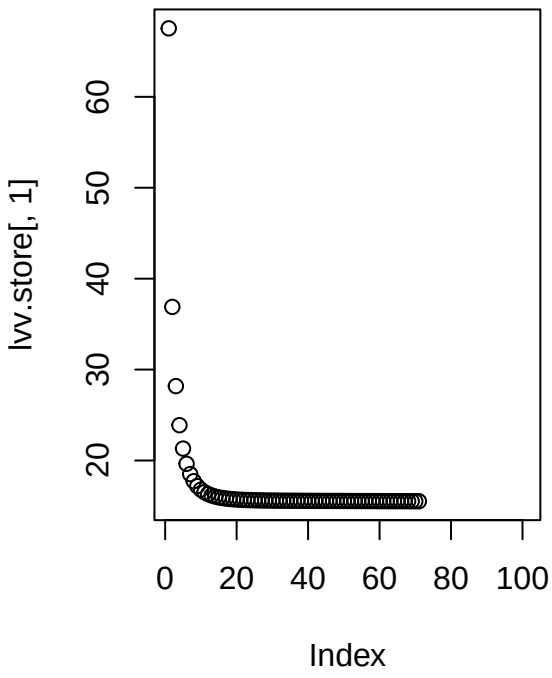
```
#Generate x
n=50
D=3
#x<-seq(-2,2,length.out=n)
x<-matrix(rnorm(n*D,0,1),nrow=n,ncol=D,byrow=FALSE)
pairs(x)
```

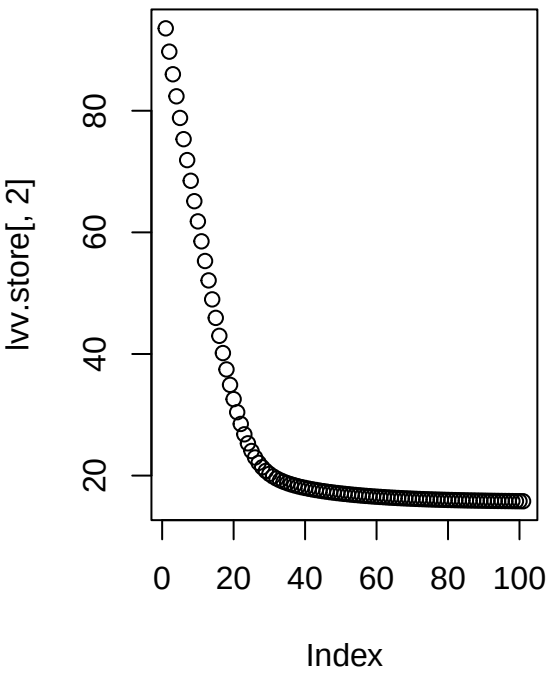



```
#Generate Covariance Matrix
length.x<-nrow(x)
K.y<-K.t(x,t1,t2)+exp(t3)*diag(1,length.x)

#Generate Y~N(0,Ky)
ev<-eigen(K.y)
L.e<-ev$values
P<-ev$vectors
D.5 = diag(sqrt(L.e))
A = P%*%D.5

mu<-c(rep(0,length.x)) #Zero Mean
set.seed(5312)
Z<-rnorm(length.x,mean=0,sd=1)
Y<-mu+A%*%Z
plot(Y)
```

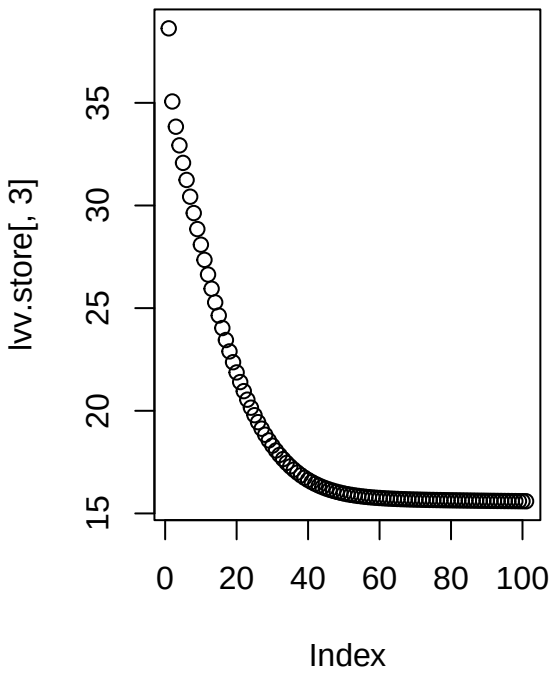


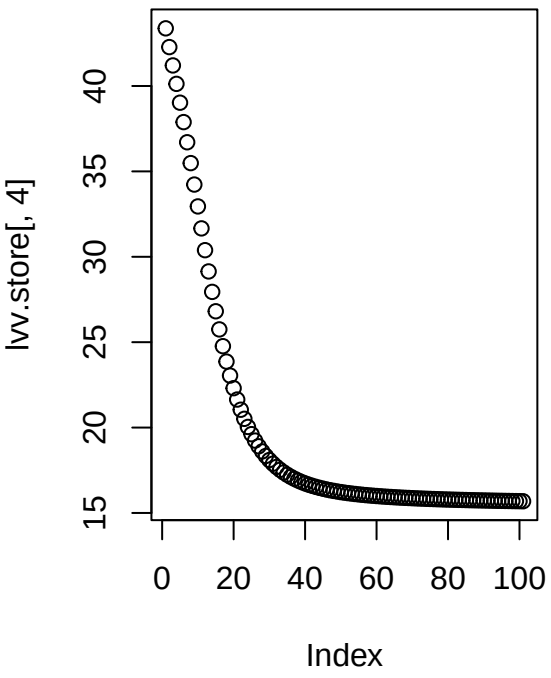

```
n.s=6
iterations<-c(70,100,100,100,100,100)
iter.max<-max(iterations)
biter.store<-matrix(rep(NA),nrow=n.s,ncol=3)
lvv.store<-matrix(rep(NA),nrow=iter.max+1,ncol=n.s)
gr.store<-matrix(rep(NA),nrow=n.s,ncol=3)
bM.store<-matrix(rep(NA),nrow=iter.max+1,ncol=(3*n.s))
theta1.0<-c(-2.8,2,2,-2.8,1.75,0.5)
theta2.0<-c(-1.5,-1.25,0.2,0.4,0.2,-2.5)
theta3.0<-c(-4,0.5,-5,-1,-9.5,0.1)
lrat=c(rep(0.01,iter.max))
for (o in 1:n.s){
  #if (o==5){lrat=c(rep(0.015,iter.max))}
result<-theta.grad(x,Y,theta1.0[o],theta2.0[o],theta3.0[o],iterations[o],lrat)
bM.store[1:(iterations[o]+1),(3*(o-1)+1):(3*(o-1)+3)]<-as.matrix(result[[1]])
gr.store[o,]<-result[[2]]
lvv.store[1:(iterations[o]+1),o]<-result[[3]]
biter.store[o,]<-bM.store[iterations[o]+1,(3*(o-1)+1):(3*(o-1)+3)]
}

colors<-c("purple","blue","red","green","orange","brown")
biter<-c(mean(biter.store[,1]),mean(biter.store[,2]),mean(biter.store[,3]))
par(mfrow=c(1,2))
plot (lvv.store[,1])
plot (lvv.store[,2])
```



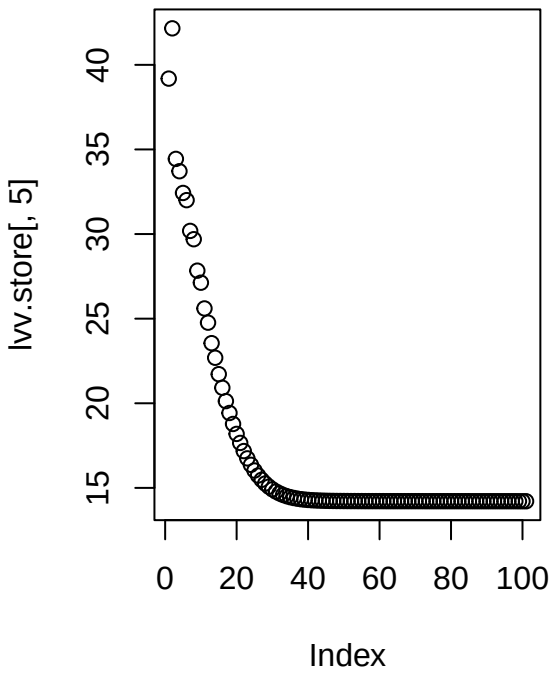


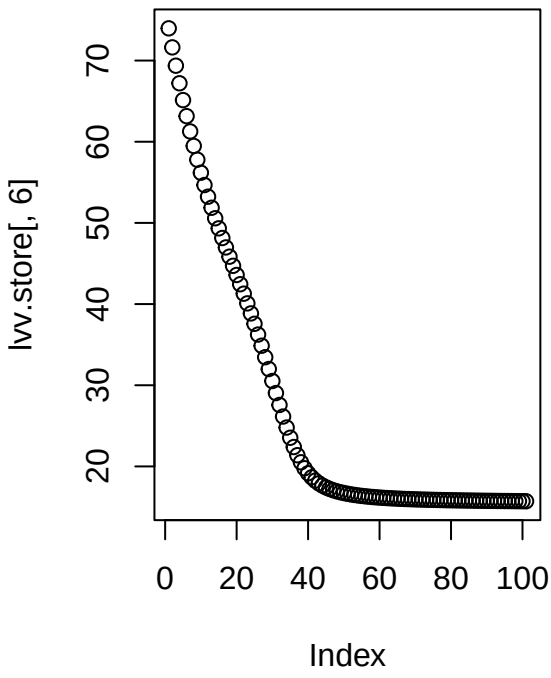

```
plot (lvv.store[,3])
plot (lvv.store[,4])
```



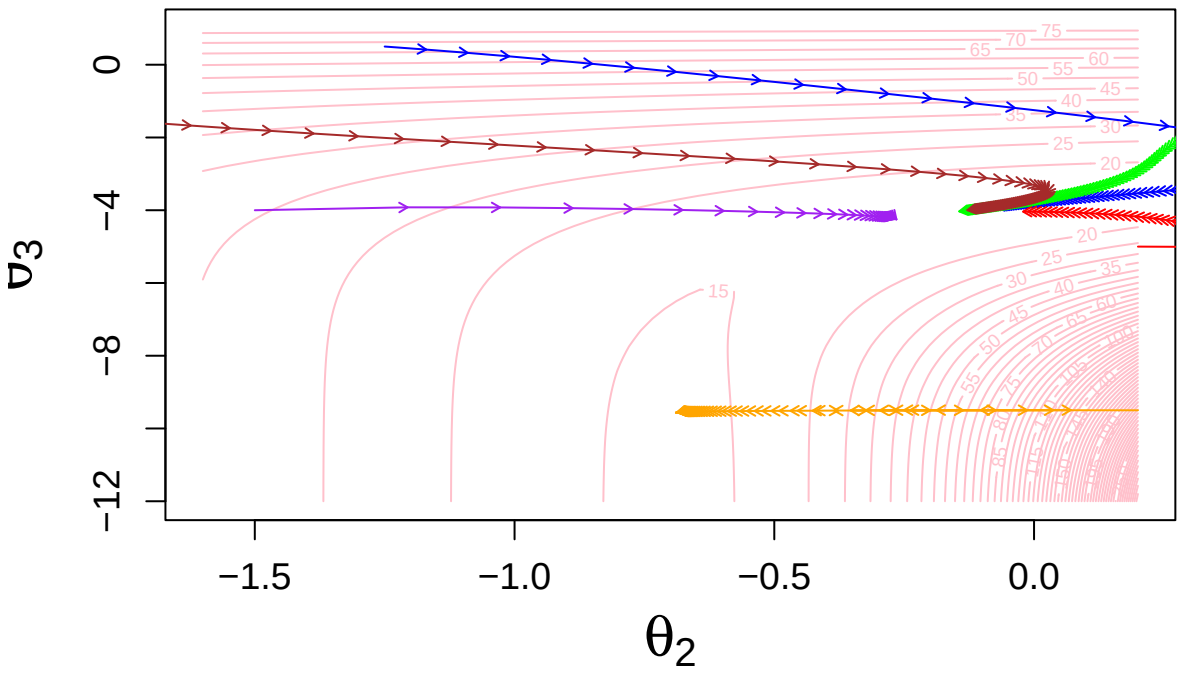



```
plot (lvv.store[,5])
plot (lvv.store[,6])
```

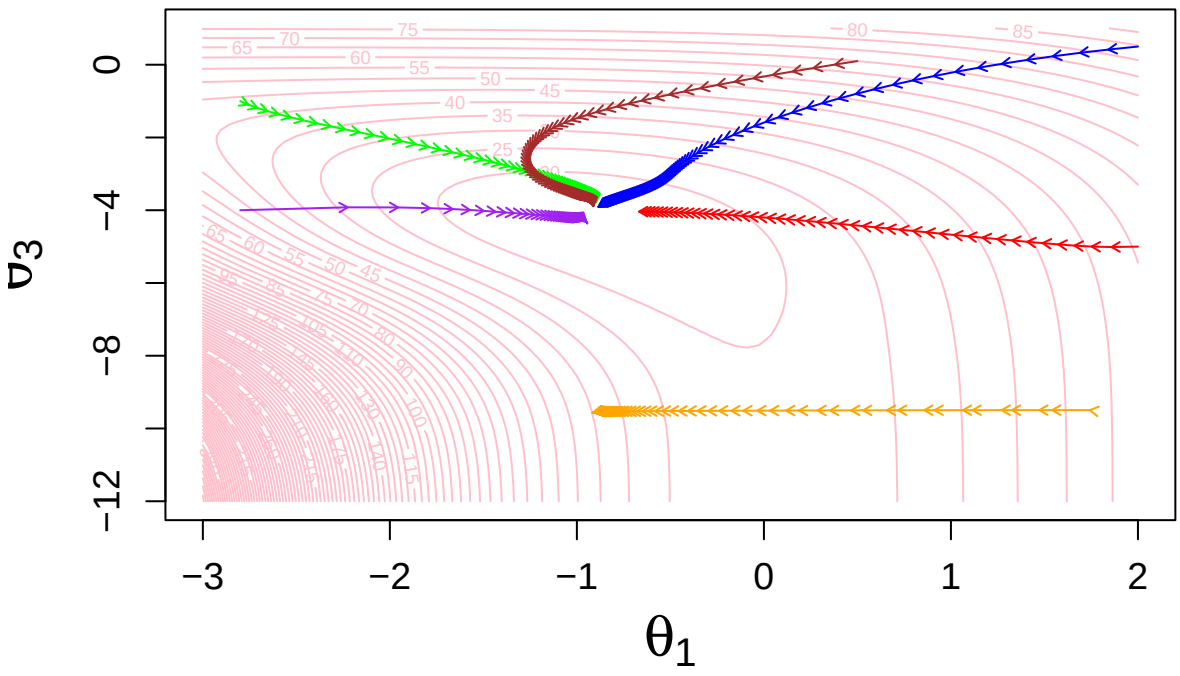




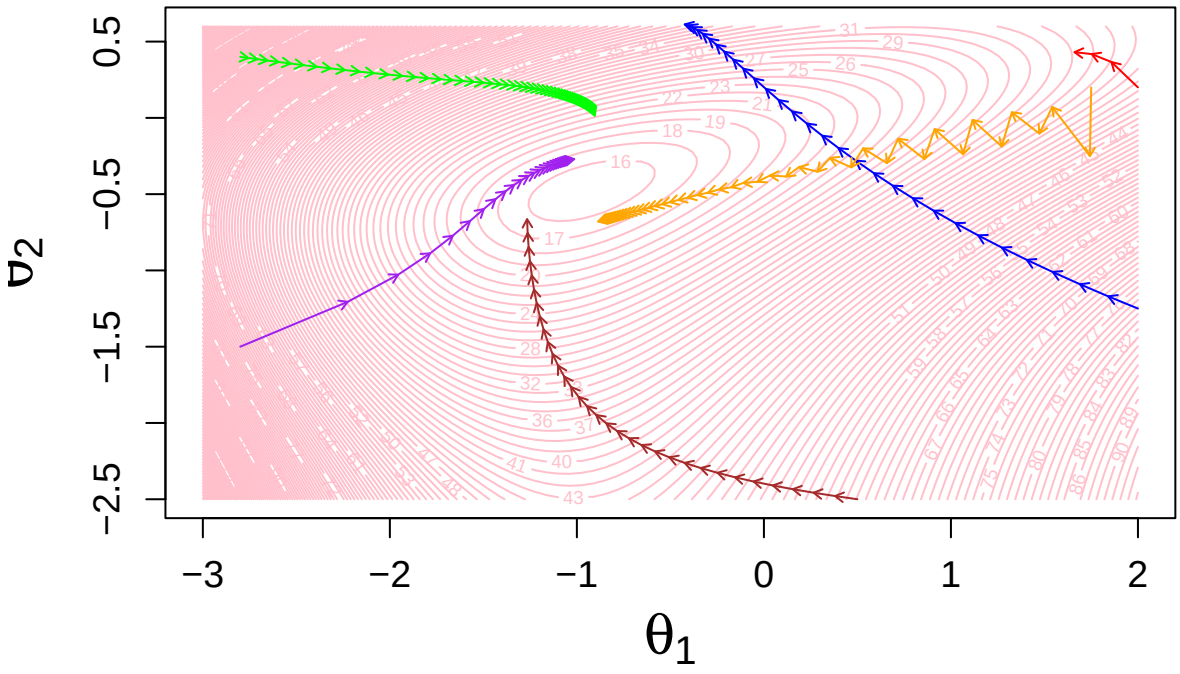

```
par(mfrow=c(1,1))
#####Fix Theta 1 = biter[1] and look at the contour
options(warn = -1)
out<-contour.plot.GP(80,-1.6,0.2,-12,1,1,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=80,drawlabels=TRUE,
     xlab=expression(theta[2]),ylab=expression(theta[3]),col="pink",
           cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM.store[(1:iterations[o]),(3*(o-1)+2):(3*(o-1)+3)]
st<-stop.num(bM3,.0001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=colors[o])}
}
```




```
#####Fix Theta 2 = biter[2] and look at the contour
out<-contour.plot.GP(80,-3,2,-12,1,2,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=60,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[3]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM.store[(1:iterations[o]),c((3*(o-1)+1),(3*(o-1)+3))]
st<-stop.num(bM3,.00012)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=colors[o])}
}
```




```
#####Fix Theta 3 = biter[3] and look at the contour
options(warn=-1)
out<-contour.plot.GP(80,-3,2,-2.5,0.6,3,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=100,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[2]),col="pink",cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM.store[(1:iterations[o]),c((3*(o-1)+1),(3*(o-1)+2))]
st<-stop.num(bM3,.0015)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=colors[o])}}
```




```
gr.store
```

```
##               [,1]          [,2]       [,3]
## [1,]  0.0004111179 -0.1189178594 -0.3359603
## [2,] -0.1981052617 -0.4050451046 -0.6786311
## [3,] -0.0534288993 -0.1805478955 -0.3848888
## [4,] -0.0620241302 -0.2604967664 -0.5314230
## [5,] -0.0001450627 -0.0001175616 -0.1077369
## [6,] -0.0489003319 -0.2766654907 -0.5791880
```

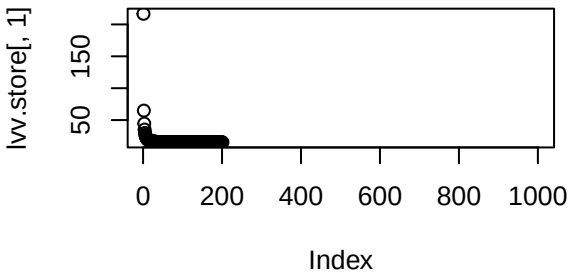
```
biter.store
```

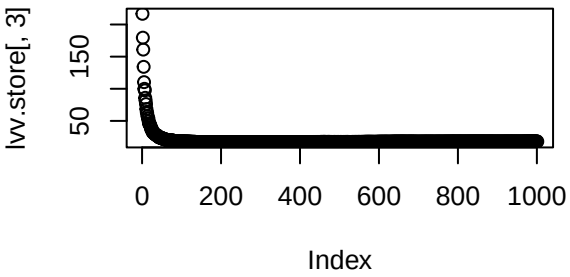
```
##            [,1]        [,2]      [,3]
## [1,] -0.9440592 -0.30100943 -4.394204
## [2,] -0.8894523 -0.08169266 -3.923292
## [3,] -0.9321058 -0.22652247 -4.212338
## [4,] -0.9189438 -0.14927675 -4.042927
## [5,] -0.9174510 -0.69013397 -9.602275
## [6,] -0.9172709 -0.13399242 -4.009543
```

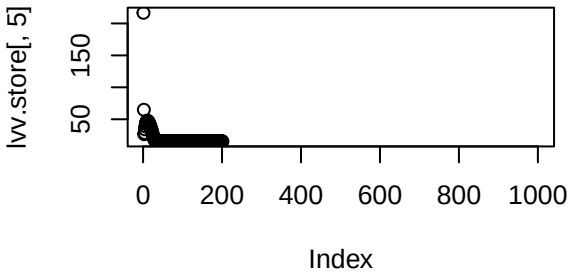
The code appears to work as they appear to converge to the trough. If the learning rate was too big they tended to diverge.

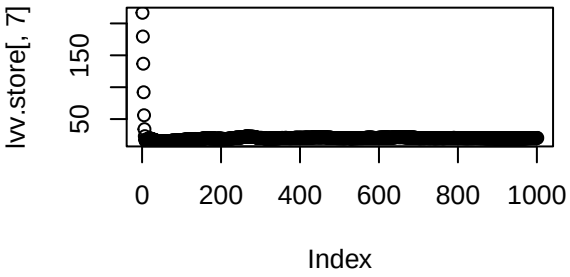
#Comparing the different methods:

```
n.s=8
#First 2 regular (green)
#Next 2 Stochastic (blue)
#Next 2 Momentum (orange)
#Last 2 Momentum and Stochastic (Red)
iterations<-c(200,200,200,200,200,200,200,200)
num.batch.setup<-c(1,1,5,5,1,1,5,5)
iter.max<-max(iterations*num.batch.setup)
biter.store<-matrix(rep(NA),nrow=n.s,ncol=3)
lvv.store<-matrix(rep(NA),nrow=iter.max+1,ncol=n.s)
gr.store<-matrix(rep(NA),nrow=n.s,ncol=3)
bM.store<-matrix(rep(NA),nrow=iter.max+1,ncol=(3*n.s))
theta1.0<-c(-2.8,0.75,-2.8,0.75,-2.8,0.75,-2.8,0.75)
theta2.0<-c(0.5,0.4,0.5,0.4,0.5,0.4,0.5,0.4)
theta3.0<-c(-6,0.1,-6,0.1,-6,0.1,-6,0.1)
mom.setup<-c(0,0,0,0,1,1,1,1)
st.setup<-c(0,0,1,1,0,0,1,1)
lrat=c(rep(0.003,iter.max))
for (t in 1:iter.max){
  lrat[t]=lrat[1]/(1+0.01*t)
}
for (o in 1:n.s){
set.seed(1234)
b<-st.setup[o]
c<-mom.setup[o]
result<-theta.grad(x,Y,theta1.0[o],theta2.0[o],theta3.0[o],iterations[o],
                   lrat,stochastic=b,num.batch=num.batch.setup[o],momentum=c,gamma=0.8)
bM.store[1:(iterations[o]*num.batch.setup[o]+1),(3*(o-1)+1):(3*(o-1)+3)]<-as.matrix(result[[1]])
gr.store[o,]<-result[[2]]
lvv.store[1:(iterations[o]*num.batch.setup[o]+1),o]<-result[[3]]
biter.store[o,]<-bM.store[iterations[o]*num.batch.setup[o]+1,(3*(o-1)+1):(3*(o-1)+3)]
}

colors<-c("green","green","blue","blue","orange","orange","red","red")
biter<-c(mean(biter.store[,1]),mean(biter.store[,2]),mean(biter.store[,3]))
par(mfrow=c(2,2))
plot (lvv.store[,1])
plot (lvv.store[,3])
plot (lvv.store[,5])
plot (lvv.store[,7])
```

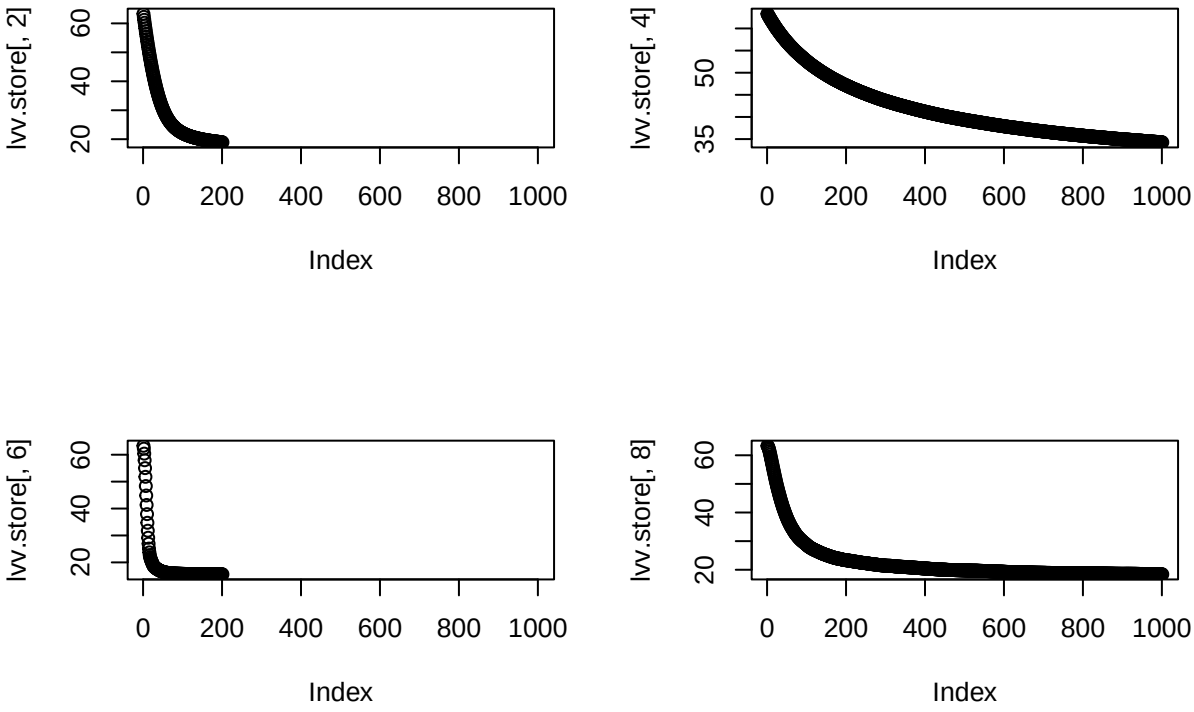




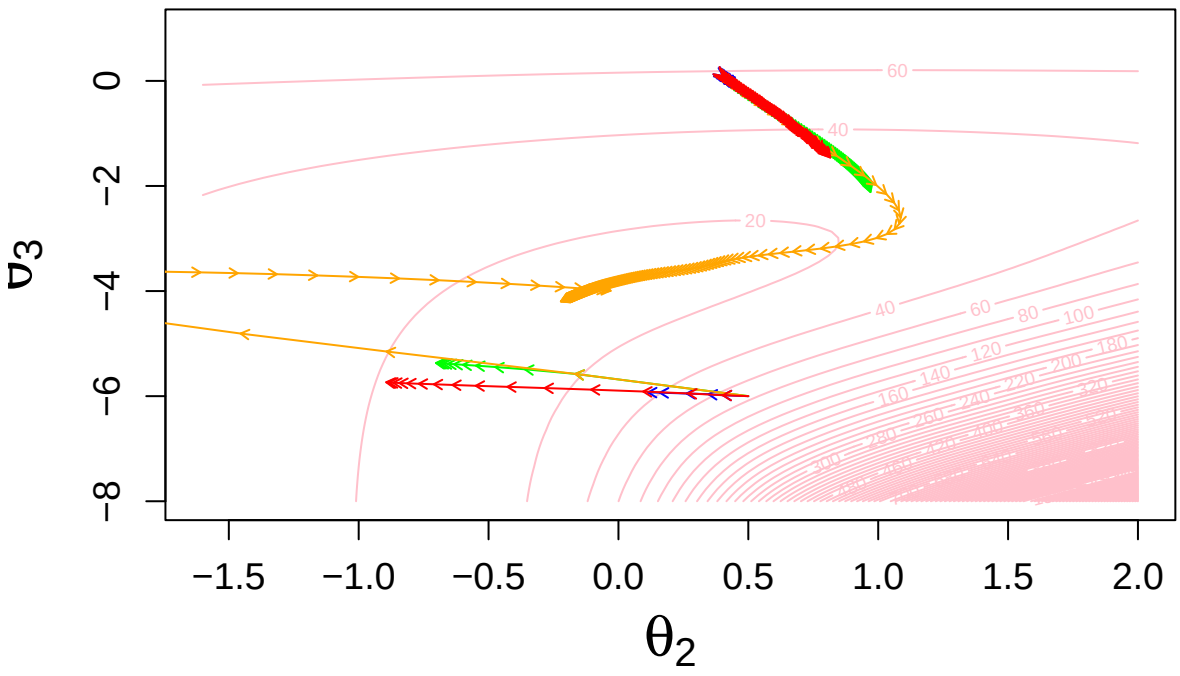




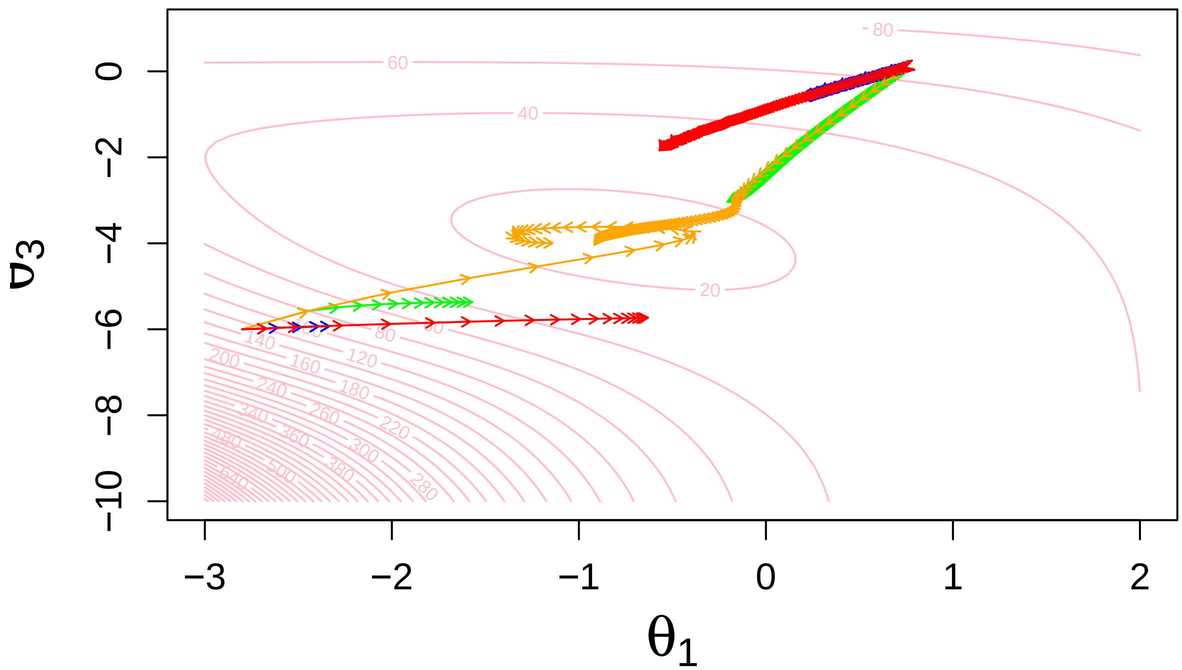

```
plot (lvv.store[,2])
plot (lvv.store[,4])
plot (lvv.store[,6])
plot (lvv.store[,8])
```



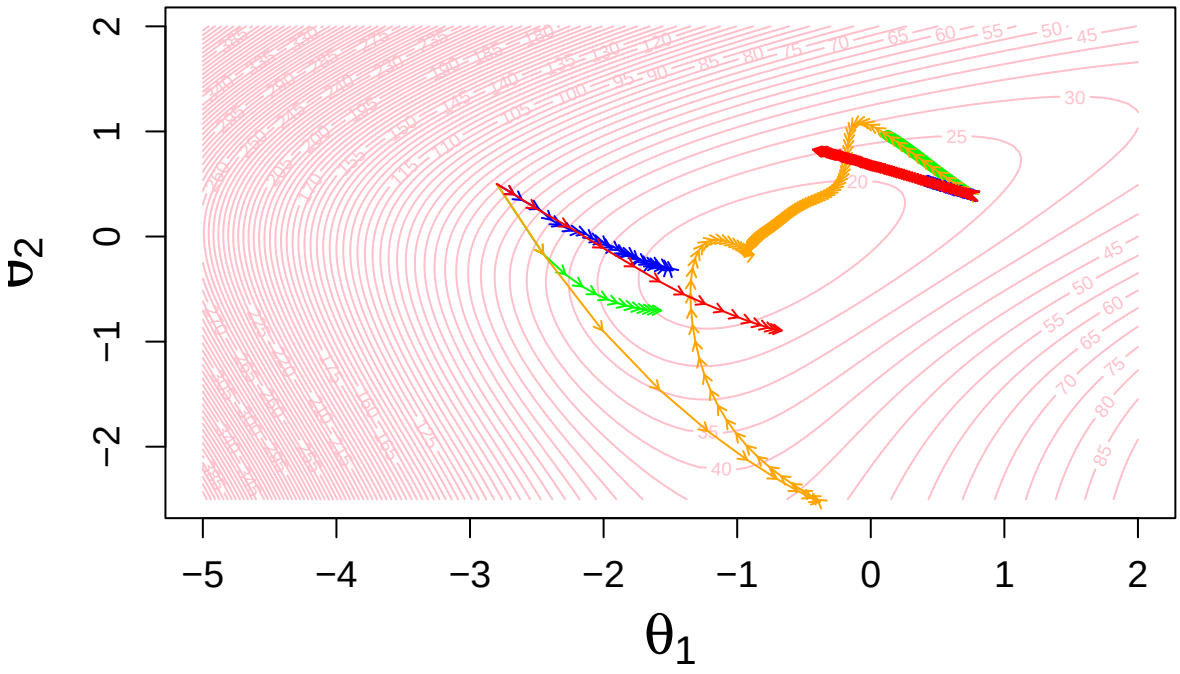

```
par(mfrow=c(1,1))
#####Fix Theta 1 = biter[1] and look at the contour
out<-contour.plot.GP(80,-1.6,2,-8,1,1,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=80,drawlabels=TRUE,
     xlab=expression(theta[2]),ylab=expression(theta[3]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM.store[(1:iterations[o]),(3*(o-1)+2):(3*(o-1)+3)]
st<-stop.num(bM3,.001)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=colors[o])}
}
```




```
#####Fix Theta 2 = biter[2] and look at the contour
out<-contour.plot.GP(80,-3,2,-10,1,2,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=40,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[3]),col="pink",
          cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM.store[(1:iterations[o]),c((3*(o-1)+1),(3*(o-1)+3))]
st<-stop.num(bM3,.0006)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=colors[o])}
}
```




```
#####Fix Theta 3 = biter[3] and look at the contour
out<-contour.plot.GP(80,-5,2,-2.5,2,3,biter,x,Y)
alga1=as.vector(out[[1]])
alga2=as.vector(out[[2]])
llmat=as.matrix(out[[3]])
contour(alga1,alga2,llmat,nlevels=100,drawlabels=TRUE,
     xlab=expression(theta[1]),ylab=expression(theta[2]),col="pink",
           cex.lab=1.8,cex.axis=1.2,lwd=1)
for (o in 1:n.s){
bM3<-bM.store[(1:iterations[o]),c((3*(o-1)+1),(3*(o-1)+2))]
st<-stop.num(bM3,.0006)
for(i in 2:(min(st))) {
   arrows(bM3[i-1,1],bM3[i-1,2],bM3[i,1],bM3[i,2],length=.05,col=colors[o])}
}
```




```
gr.store
```

```
##              [,1]        [,2]       [,3]
## [1,]  0.185550105  0.02952356 -0.3033309
## [2,] -1.485432590 -2.04614064 -2.8860354
## [3,]  1.294680742  1.71277014 -0.1116096
## [4,]  0.165846772 -0.43630049 -1.5512276
## [5,] -0.006158505 -0.11413294 -0.3290126
## [6,] -0.018680177 -0.15042425 -0.3660954
## [7,]  0.799171769  1.61332789 -0.1455497
## [8,]  1.454171039  0.04243149 -0.7790798
```

```
biter.store
```

```
##            [,1]       [,2]      [,3]
## [1,] -0.9514284 -0.5462126 -5.460001
## [2,] -0.2087371  0.7146394 -3.041681
## [3,] -0.7841217 -0.2820974 -5.853446
## [4,] -0.2931890  0.7899462 -1.338355
## [5,] -0.9443533 -0.3181999 -4.443292
## [6,] -0.9382007 -0.2505753 -4.265779
## [7,] -0.6825135 -0.1443168 -5.700790
## [8,] -0.7054245  0.3160531 -2.942596
```

```
sqrt(exp(biter.store))
```

```
##           [,1]      [,2]       [,3]
## [1,] 0.6214411 0.7610119 0.06521927
## [2,] 0.9008932 1.4294928 0.21852819
## [3,] 0.6756630 0.8684470 0.05357230
## [4,] 0.8636441 1.4843443 0.51212960
## [5,] 0.6236433 0.8529111 0.10843047
## [6,] 0.6255648 0.8822431 0.11849439
## [7,] 0.7108764 0.9303835 0.05782149
## [8,] 0.7027794 1.1711973 0.22962722
```

```
cbind(t1,mean(((biter.store[,1]))))
```

```
##             t1
## [1,] -1.386294 -0.688496
```

```
cbind(t2,mean(((biter.store[,2]))))
```

```
##              t2
## [1,] -0.7133499 0.03490459
```

```
cbind(t3,mean(((biter.store[,3]))))
```

```
##            t3
## [1,] -9.21034 -4.130743
```

With momentum and too large a learning rate we can see that it might diverge (that is true in general if the learning rate is too large.)

#Splitting the data into training and testing data and using to estimate f(x)

Using the best of the descents above (values of original parameters) were: [5,] 1.0636978 0.8007951 0.01111188 [6,] 1.0809990 0.8063751 0.01037233 So we used the estimates for

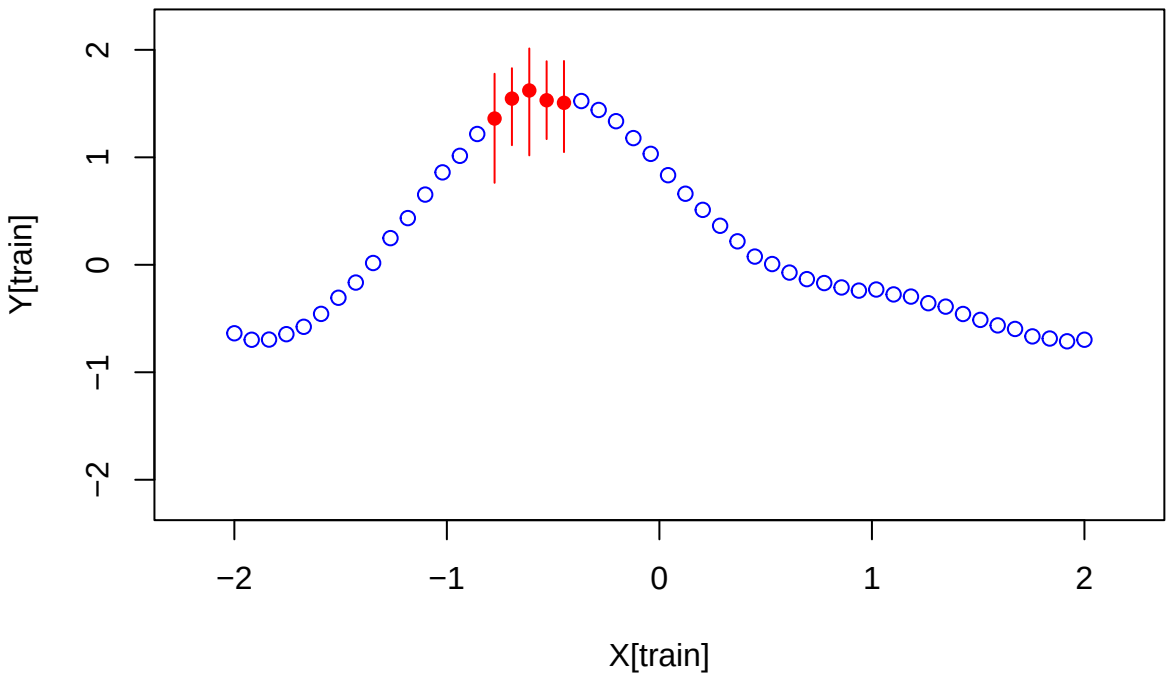
$$\theta$$

as:

$$\theta_1 = log(1.07^2)$$
$$\theta_2 = log(0.81^2)$$
$$\theta_3 = log(0.01^2)$$

```
#set.seed(12543)
X<-as.matrix(x.save)
Y<-as.matrix(y.save)
t1<-log(2.07^2)
t2<-log(0.81^2)
t3<-log(0.3^2)

#####################################################################
train<-c(1:15,21:50)
```

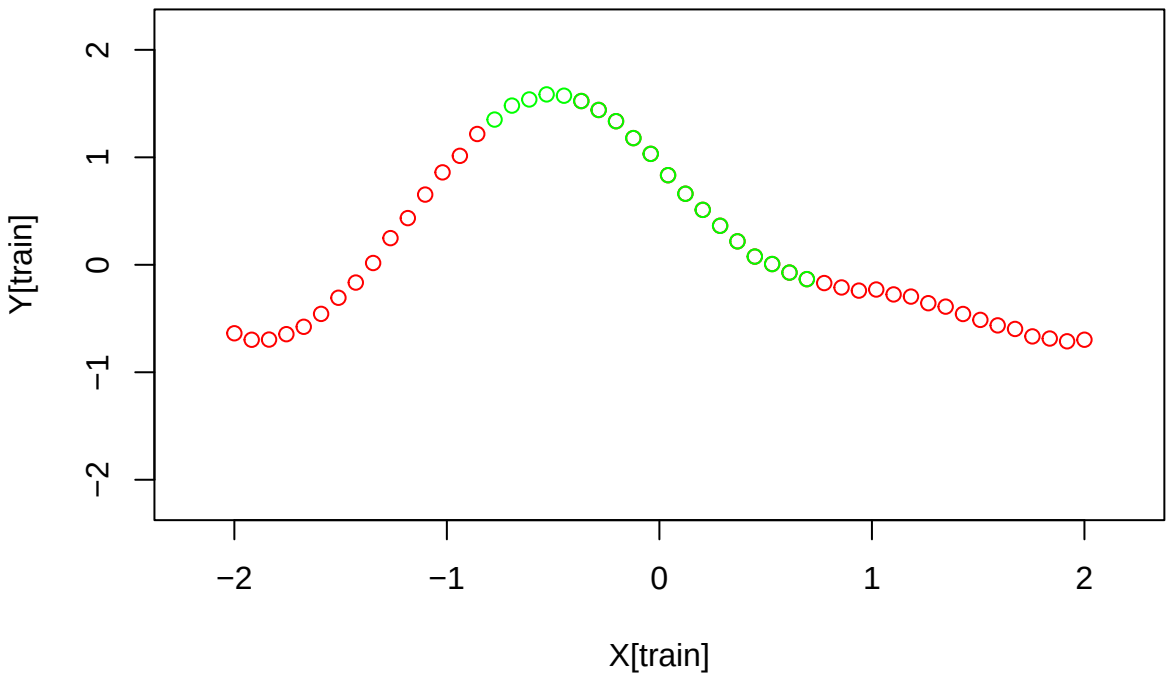
```
test<-c(16:20)
K<-K.t(X,t1,t2)+exp(t3)*diag(length(Y))
Kxx<-K[train,train]
Kxy<-K[train,test]
Kyx<-K[test,train]
Kyy<-K[test,test]
ev=eigen(Kxx)
Lx<-ev$values
Qx<-ev$vectors
Kxinv<-Qx%*%diag(1/Lx)%*%solve(Qx)
B=Kyx%*%Kxinv
Mu<-B%*%Y[train] #Mean of postieror of f*
Sig<-Kyy-B%*%Kxy #Covariance matrix of posterior of f*
L<-chol(Sig)

#####################################
set.seed(1542)
nd=50
draws<-matrix(NA,ncol=nd,nrow=length(test))   #f*~N(Mu,Sig)
for (i in 1:nd){
  draws[,i]<-Mu+L%*%c(rnorm(length(test),0,1)) #Record each draw in a column
}

### plot GP inference
plot(X[train],Y[train],col="blue",xlim=c(-2.2,2.2),ylim=c(-2.2,2.2))
qmat = apply(t(draws),2,quantile,probs=c(.1,.5,.9))
for(i in 1:nrow(draws)) {
    ii = test[i]
    lines(X[rep(ii,2)],qmat[c(1,3),i],col='red')
    points(X[ii],qmat[2,i],col="red",pch=16)
}
```




```
# lines(X[train],Y[train],col='gray',lwd=3,type="p")
```

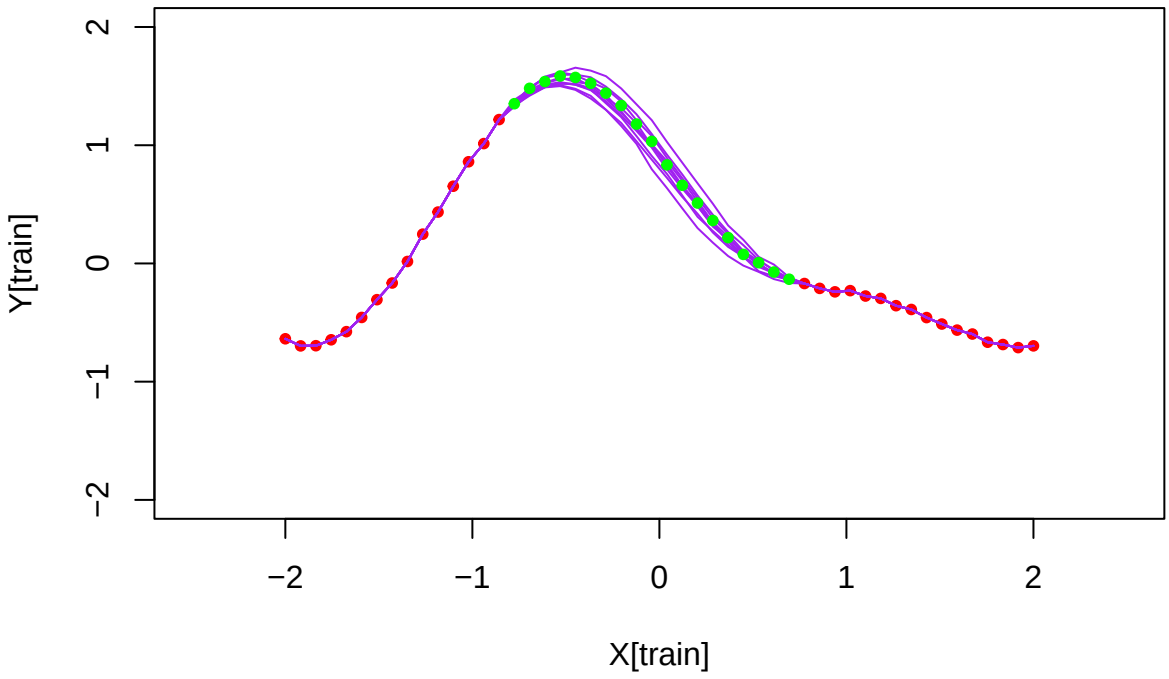
The red vertical line segments show the 80% credible interval for what the values of f*(x_i) are for the x_i in the testing set.

```
set.seed(12543)
X<-as.matrix(x.save)
Y<-as.matrix(y.save)
#plot(X,Y)
#TrSel<-c(1:nrow(X))
#train <- sample(TrSel,15,replace=FALSE)
plot(X[train],Y[train],col="red",xlim=c(-2.2,2.2),ylim=c(-2.2,2.2))
#test <- setdiff(TrSel,train)
train<-c(1:15,35:50)
test<-c(16:34)
points(X[test],Y[test],col="green")
```




```
t1<-log(1.07^2)
t2<-log(0.81^2)
t3<-log(0.01^2)
K<-K.t(X,t1,t2)+exp(t3)*diag(1,nrow(X))

S = svd(K[train,train])    #Singular Value Decomposition on the training data.
#S = UDV^T, D = Diagonal, U, V are orthonormal matrices. (Note U^-1=U^T, V^-1=V^T)
A = K[,train]%*%S$u%*%diag(1/S$d)%*%t(S$v)
#K*(K^T*K)^1 = (UDV^T)(VDU^T*UDV^T)^-1 =
#(UDV^T)(VDDV^T)^-1 =
#UDV^(T)VD^(-1)D^(-1)V^T = UD*(V^(T)V)*D^(-1)D^(-1)V^T = UD^-1V^T
#So A = K*(K^T*K)^1
#on th eblock of K associated with the training data.

V = K - A%*%K[train,]
L = svd(V)
L = L$u%*%diag(sqrt(L$d))

m = A%*%Y[train]
ts = sort(X,index.return=TRUE)

plot(X[train],Y[train],col='red',type='p',pch=20,cex = 1,xlim=c(-2.5,2.5),ylim=c(-2,2))
####Draws from the posterior for f:
for (h in 1:10){
  Z <- m + L%*%rnorm(50,0,1)
  lines(X[ts$ix],Z[ts$ix],col='purple')
```

```
}
lines(X[test],Y[test],col='green',type='p',pch=20,cex = 1)
```




```
##############################################################
```

The purple lines show some draws,

$$f^*$$

, defined on the real line. The green dots show the values

$$f^*(x_i)$$

at the test values (x_i).