Stochastic Gradient Descent

Su Chen

September 17, 2016

function to calculate negative likelihood function

```
Neg_ll = function(x, y, b, m)
{
  xb = x %*% b
  Neg_ll = sum(m*log(1+exp(- xb))) + sum((m-y)*xb) #used simplified version
  return (Neg_ll)
}
```

function to calculate Gradient of negative likelihood function

```
Gradient_Cal = function(x, y, beta, m)
{
  w = 1/(1+exp(- (x %*% beta))) #saving w as a vector instead of diag matrix
  Gradient = t(x) %*% (m*w - y) #use m*w here to save computation time
  return (Gradient)
}
```

function to calculate Robbins-Monro step size

```
RM_Step_Cal = function(C, t, t0, alpha)
{
  Step = C*(t+t0)^(-alpha)
  return (Step)
}
```

function of stochastic gradient decsent with decaying step size by Robbins-Monro rule

Note: Sto_Gradient_Desc is the function of stochastic gradient decsent with constant step size, the R code is almost identical so it's hidden in the output.

```
Sto_Gradient_Desc_RM = function(x, y, m, precision, C, t0, alpha, lambda, iter_max)
{
  N = nrow(x)
  P = ncol(x)
  #initialize all the variables
  beta = matrix(0, P, iter_max)
  nll_ini = Neg_ll(x, y, rep(0,P), m)
  nll = rep(nll_ini, iter_max)
  nll_avg = rep(nll_ini, iter_max)
  nll_ex_avg = rep(nll_ini, iter_max)
```

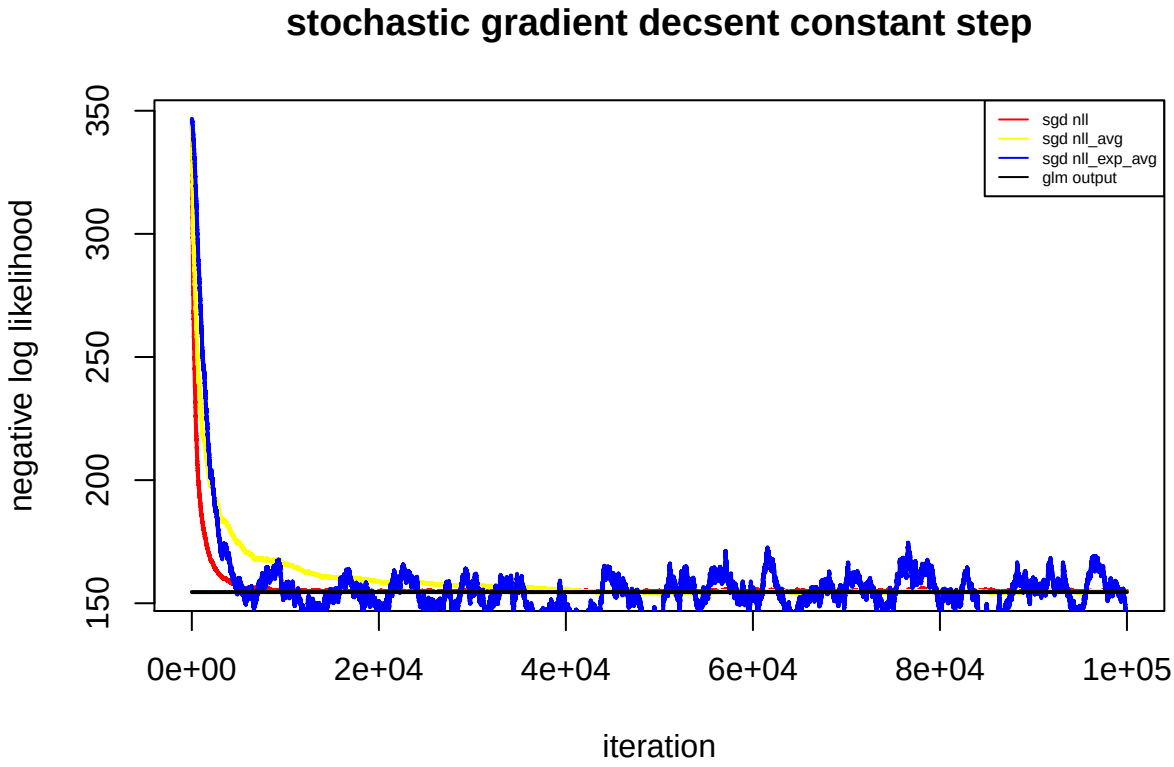
```
  delta = 1
  iter = 1

  while ( (iter < iter_max) && (delta > precision) )
  {
    #step=Step[iter]
    s = RM_Step_Cal(C, iter, t0, alpha)
    index = sample(1:N, 1) #sample 1 data point from the whole data set
    xi = x[index,,drop=F]
    yi = y[index]
    mi = m[index,]
    g = Gradient_Cal(xi, yi, beta[,iter], mi)
    beta[,iter+1] = beta[,iter] - s*g #update beta
    delta = sqrt(sum((beta[,iter+1] - beta[,iter])^2))/ sqrt(sum(beta[,iter]^2)) #calculat absolute err
    nll[iter+1] = Neg_ll(x, y, beta[,iter+1], m) #calculate neg log likelihood for whole data set
    nll_xi = Neg_ll(xi, yi, beta[,iter+1], mi)*N #calculate neg log likelihood for single data sacle to
    nll_avg[iter+1] = (nll_xi + iter*nll_avg[iter])/(iter+1)
    nll_ex_avg[iter+1] = lambda*nll_xi + (1-lambda)*nll_ex_avg[iter]
    iter = iter + 1 #keep track of iteration
  }
  return (list(iter, beta[, 1:iter], nll[1:iter], nll_avg[1:iter], nll_ex_avg[1:iter]))
}
```

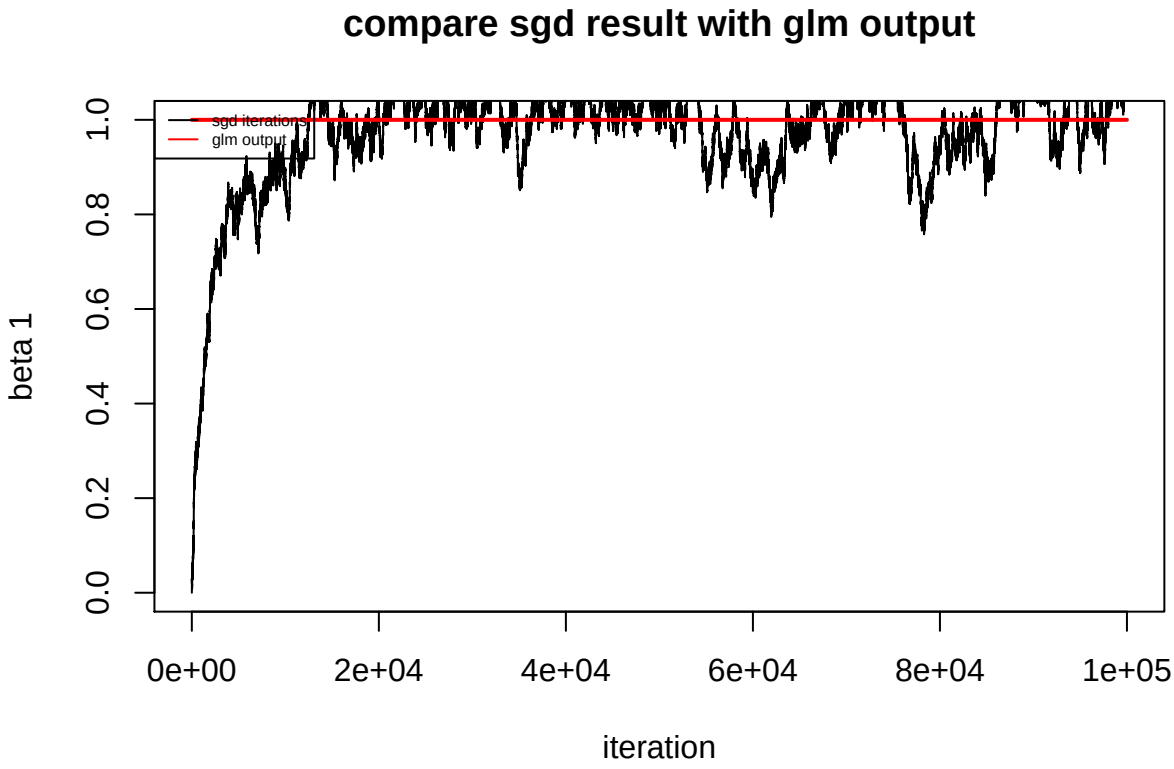
Simulate data and run stochastic gradient descent

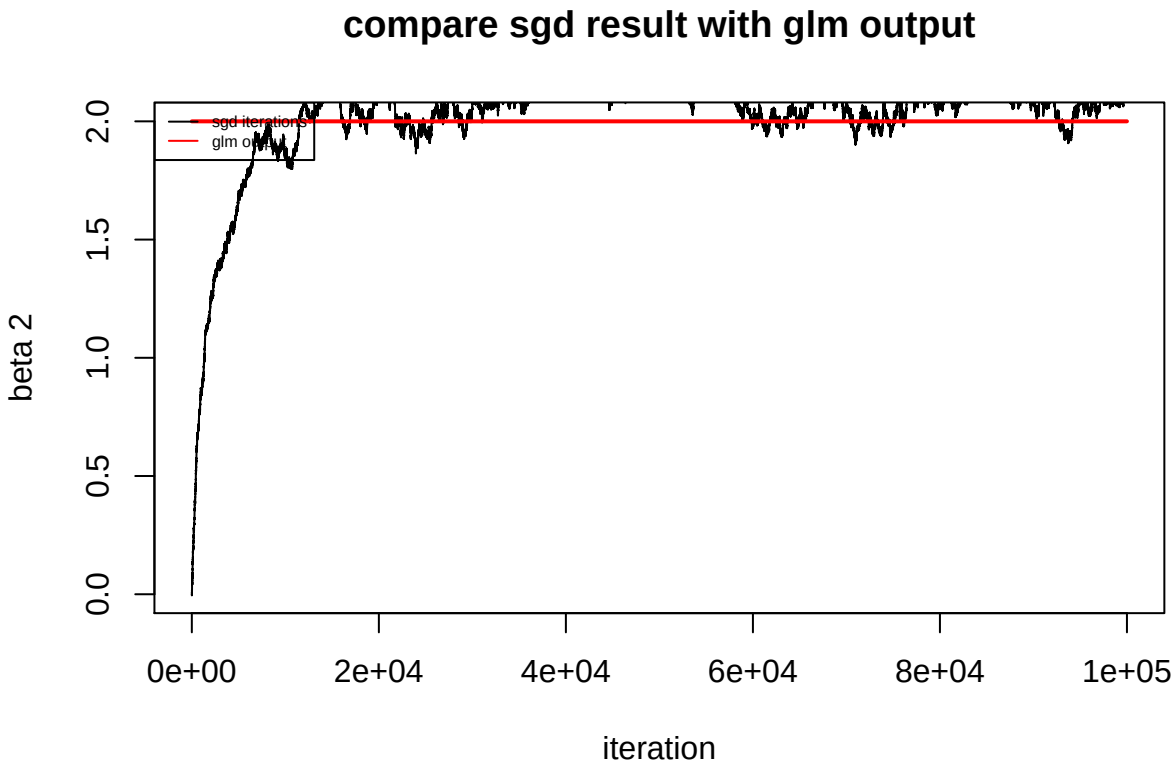
```
N = 500
P = 3
X_sim = as.matrix(cbind(rep(1,N), rnorm(N), rnorm(N)))
beta_sim = c(1, 2, -3)
W_sim = 1/(1+exp(-X_sim %*% beta_sim))  # pass through an inv-logit function
Y_sim = matrix(rbinom(N,1,W_sim), N, 1)
M_sim = matrix(1, N, 1)

#run glm and save output for comparision later
glm_sim = glm(Y_sim~X_sim-1, family='binomial')
result_glm = as.vector(glm_sim$coefficients)
nll_glm = Neg_ll(X_sim, Y_sim, result_glm, M_sim)
```

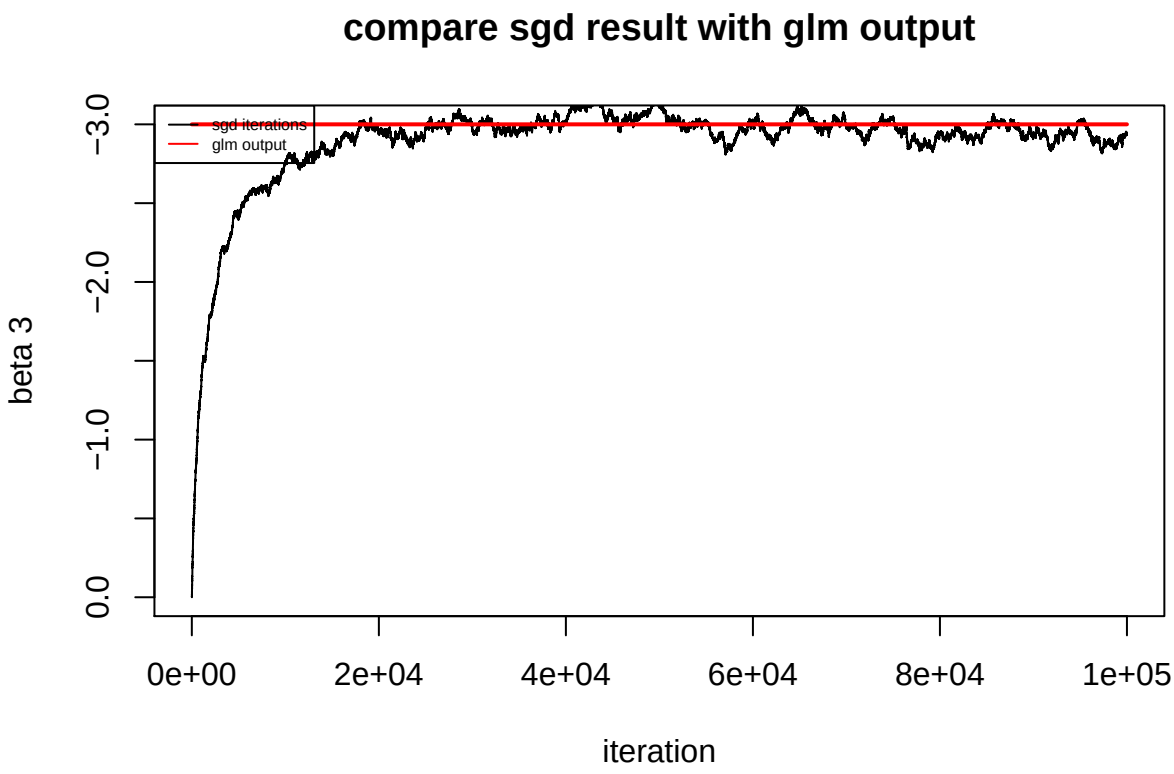
```
#stochastic gradient decsent with constant step size
result_sgd = Sto_Gradient_Desc(x=X_sim, y=Y_sim, m=M_sim,
                               precision=1E-10, step= 0.01,
                               lambda = 0.001, iter_max=100000)
```

plot likelihood and betas for SGD constant step



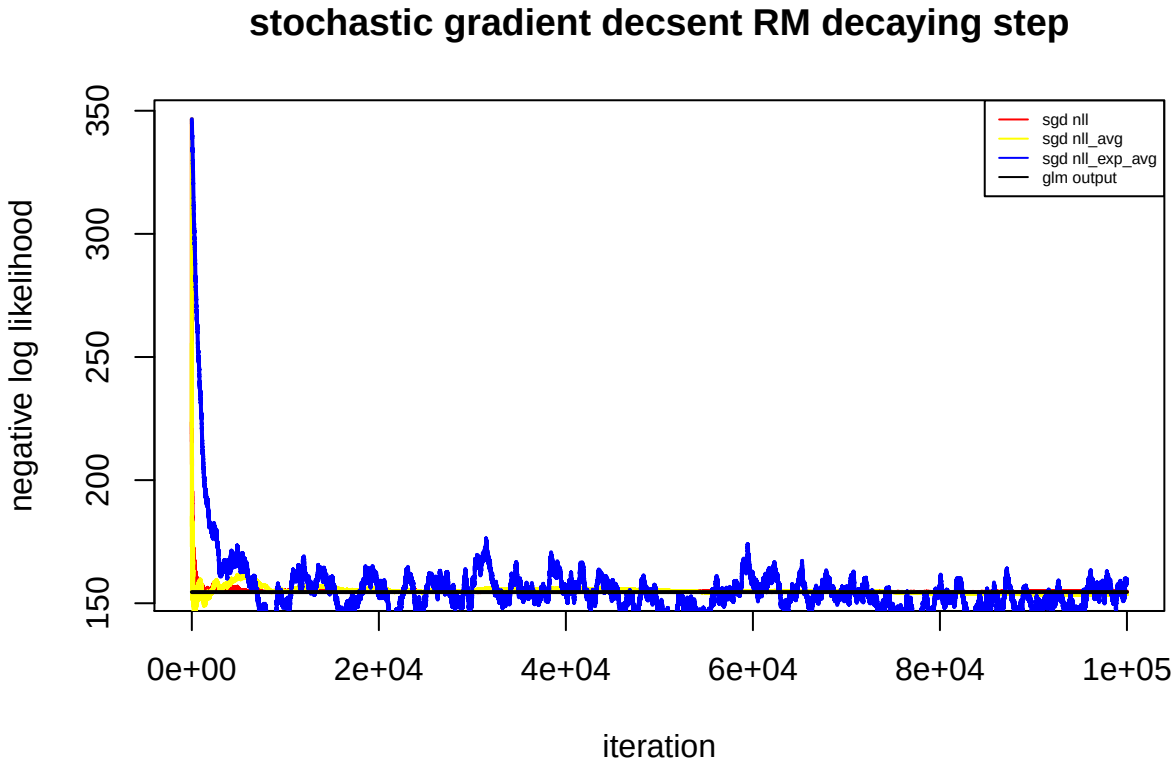


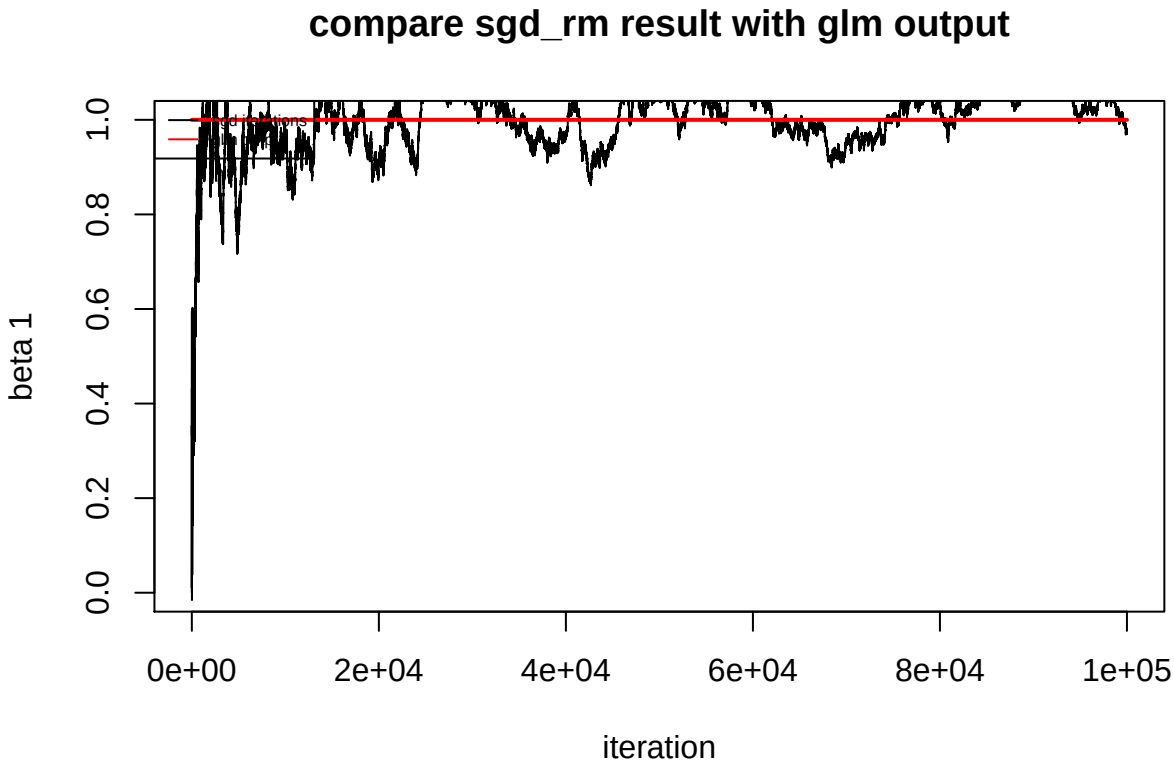

compare sgd result with glm output
sgd iterations
glm output
beta 2
0.0
0.5
1.0
1.5
2.0
0e+00
2e+04
4e+04
6e+04
8e+04
1e+05
iteration

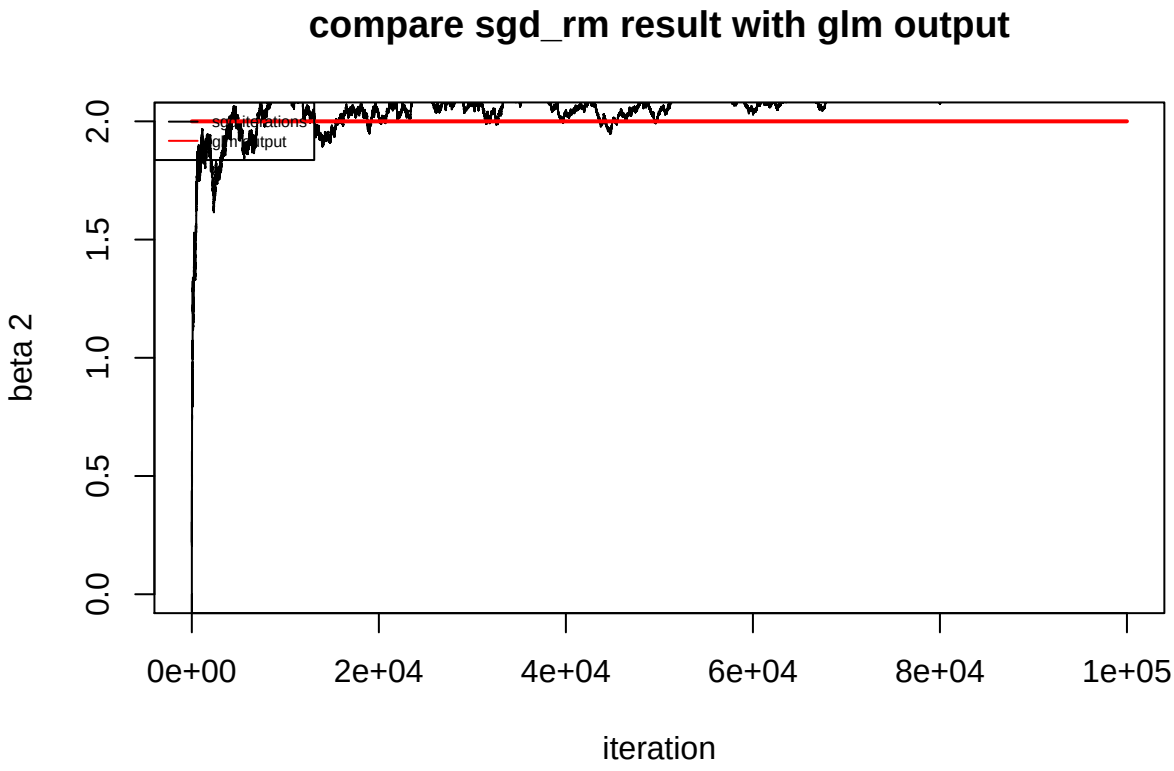


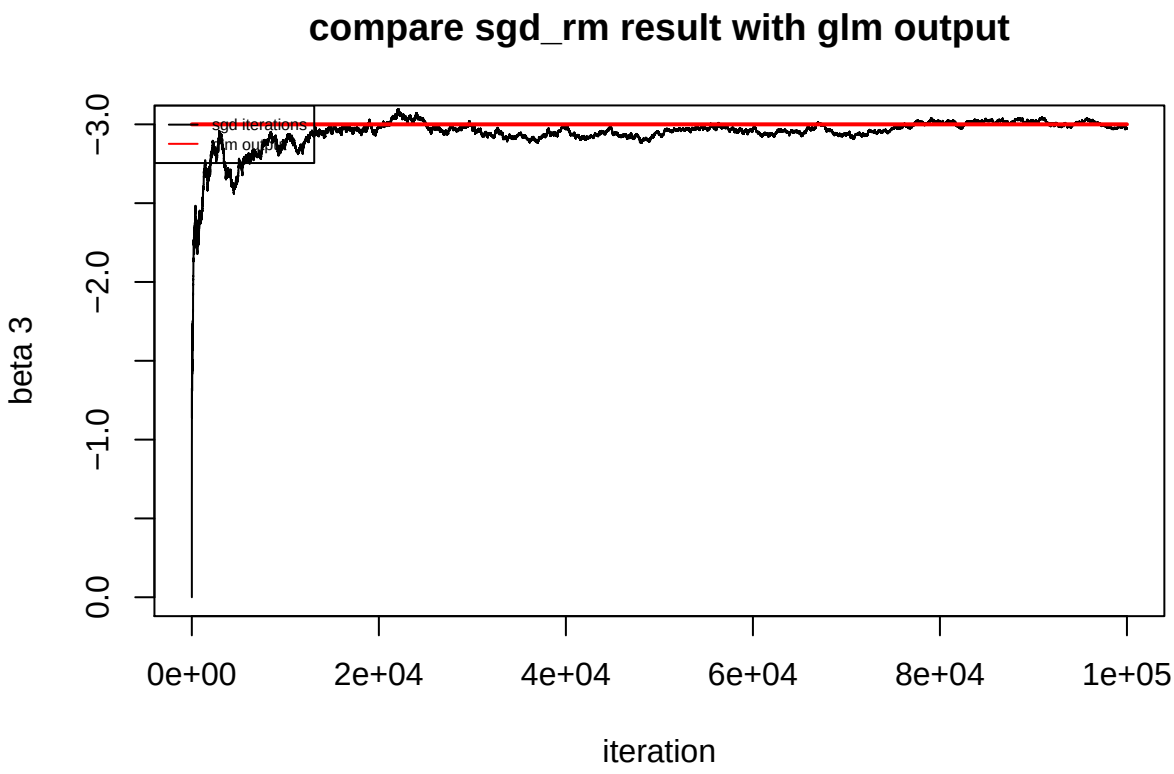

```
#stochastic gradient decsent with RM decayig step size
result_sgd_rm = Sto_Gradient_Desc_RM(x=X_sim, y=Y_sim, m=M_sim,
                                     precision=1E-10, C=1, t0=1,      alpha=0.5,
                                     lambda = 0.001, iter_max=100000)
```

plot likelihood and betas for SGD decaying step






compare sgd_rm result with glm output
sgd_iterations
glm_output
beta 2
0.0
0.5
1.0
1.5
2.0
0e+00
2e+04
4e+04
6e+04
8e+04
1e+05
iteration



compare final beta with output from glm and sgd

```
##       true beta       glm       SGD     SGD_RM
## beta1         1  1.007946  1.096830  0.9731861
## beta2         2  2.074633  2.116589  2.1561653
## beta3        -3 -2.970701 -2.948648 -2.9745075
```