Better Gradient Descent

I simulated some silly data to run the following algorithm.

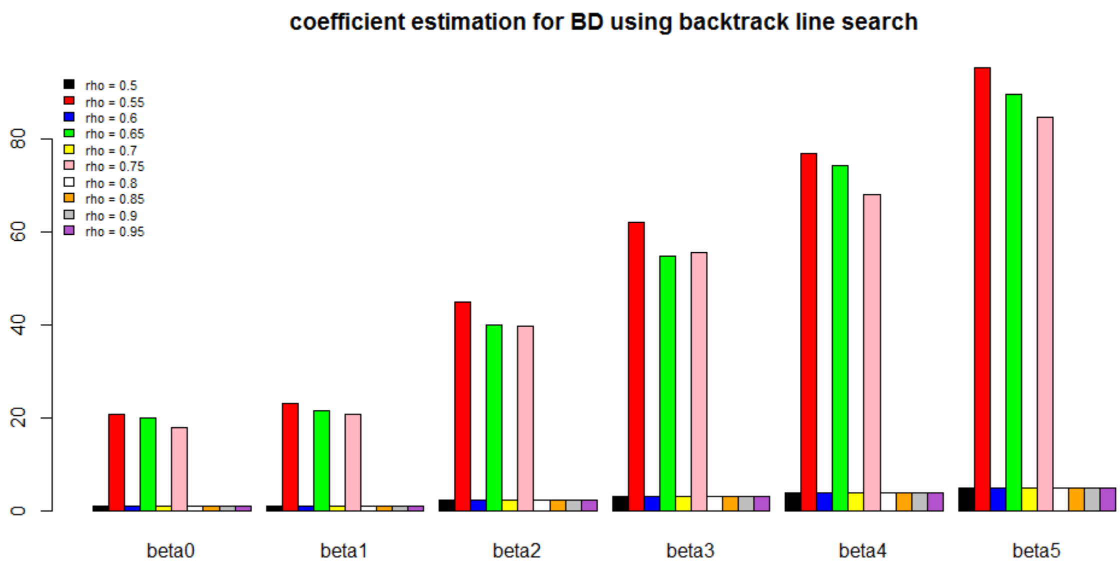
```
set.seed(128)
n = 1000
x1 = rnorm(n)
x2 = rnorm(n)
x3 = rnorm(n)
x4 = rnorm(n)
x5 = rnorm(n)
z = 1 + 1 * x1 + 2 * x2 + 3 * x3 + 4 * x4 + 5 * x5
pr = 1/(1+exp(-z))
y = rbinom(n,1,pr)
```

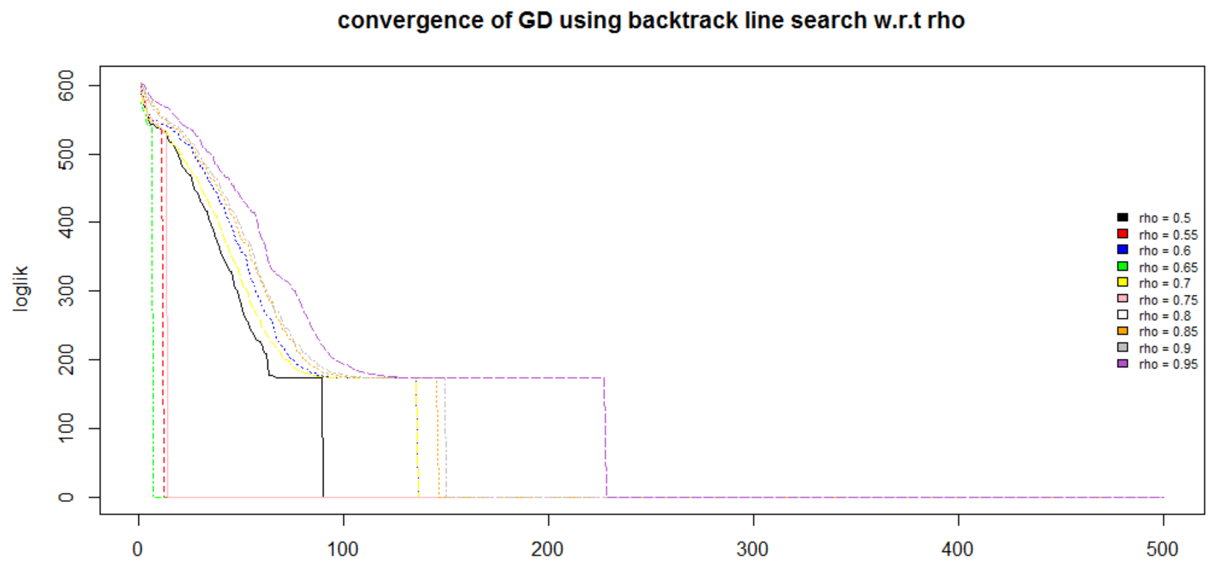
Backtrack line search

Performance with respect to rho (c = 0.001, alpha0 = 1)

The estimation does not work well for rho = 0.55, rho = 0.65, and rho = 0.75. From the log likelihood value, the problem is that algorithms with those parameter values converge fast. Any idea why this is happening? I use log likelihood value as convergence criterion.

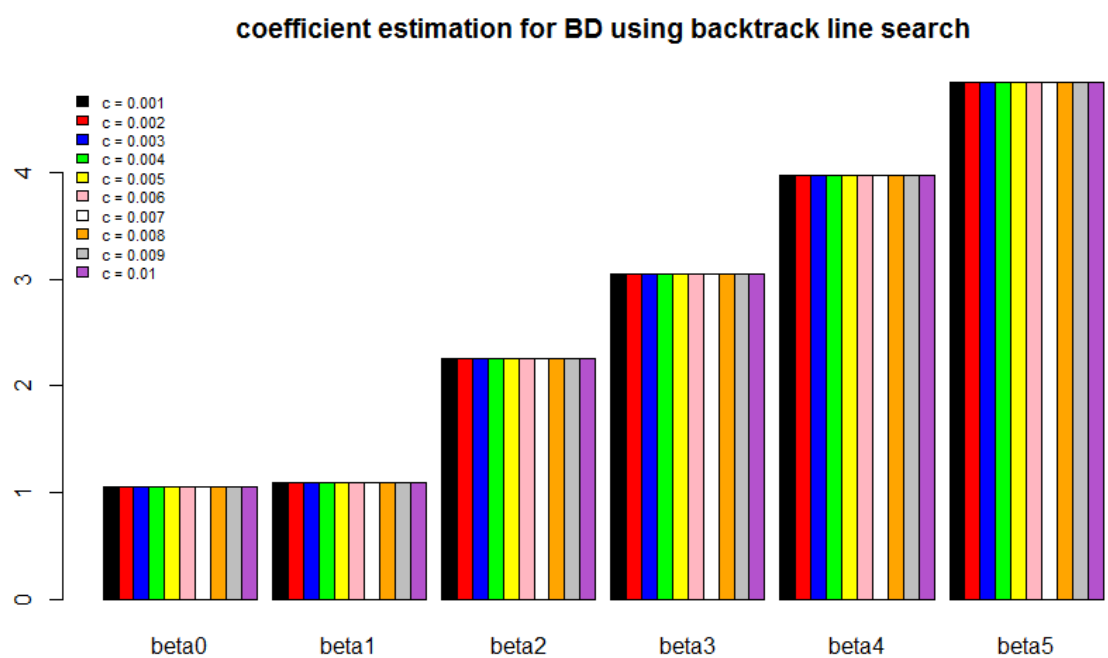
As rho grows, it takes more iterations to converge. That makes sense since a big rho leads to small change in step size, thus taking more steps to get there.

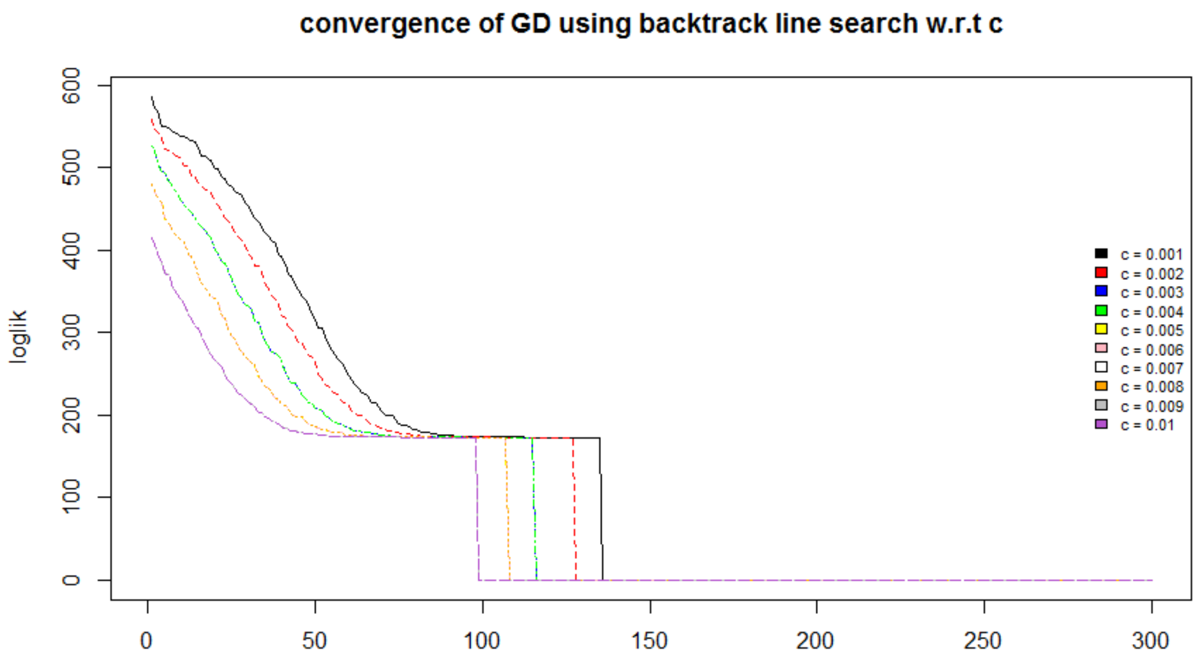




Performance with respect to c (rho = 0.7, alpha0 = 1)

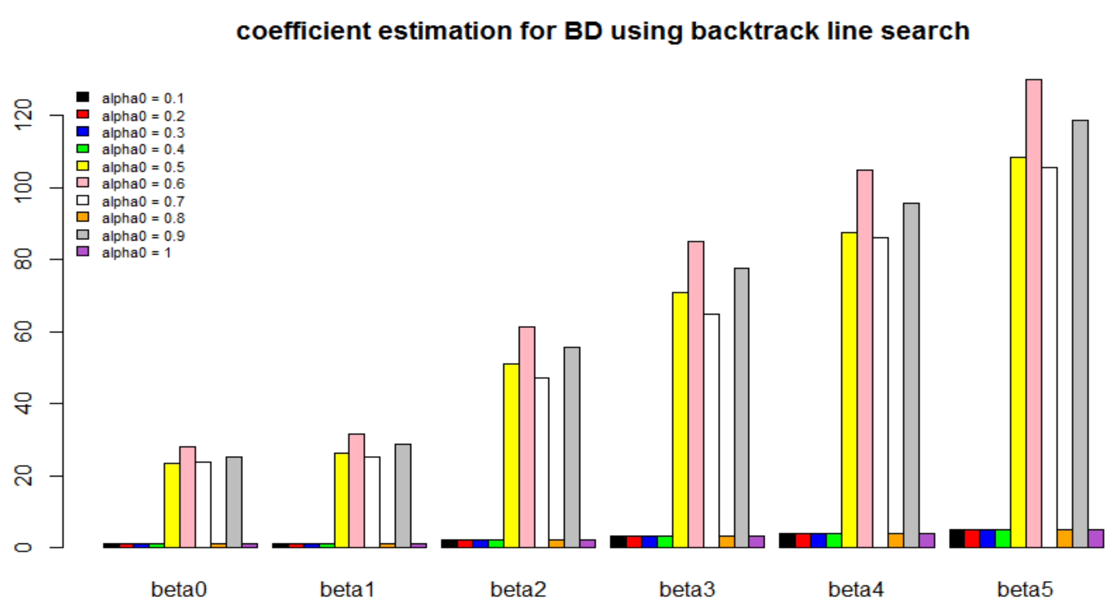
The estimation works well. As c grows, it takes fewer iterations to converge. Again, a small c leads to small change in step size in each iteration, thus taking more steps to get there.

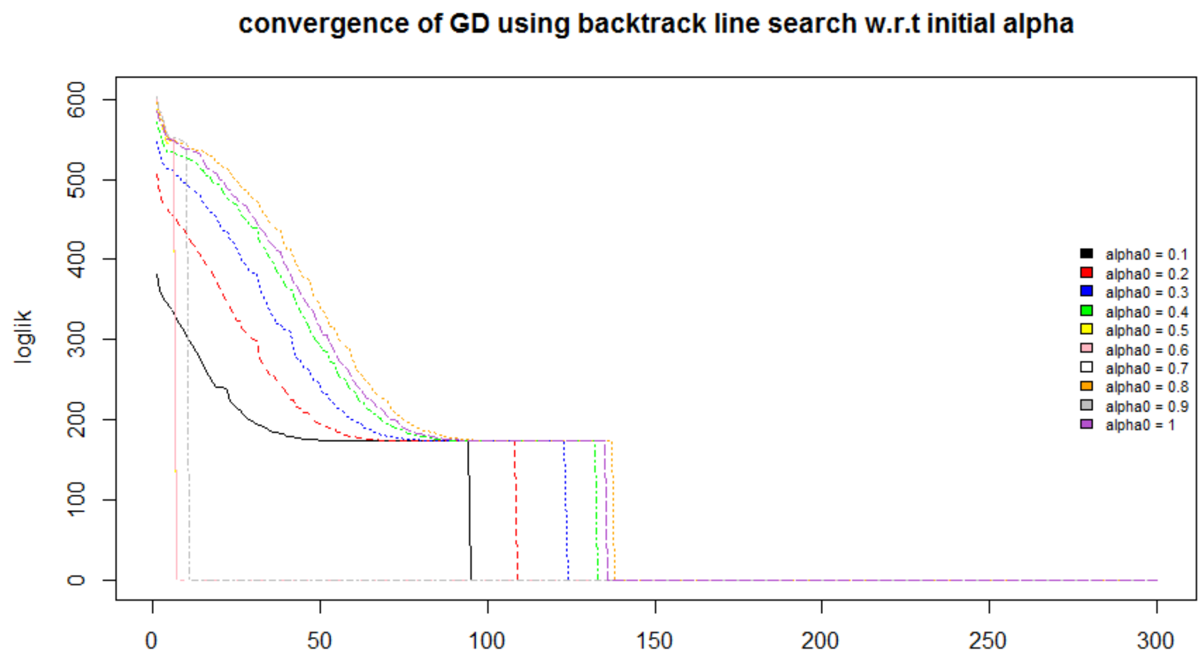




Performance with respect to c (rho = 0.7, c = 0.001)

The estimation breaks again for alpha0 = 0.5, 0.6, 0.6, 0.9. The problem is the same as before that it converges too fast. Any idea why this is happening? As alpha0 grows, it takes more iterations to converge. Since it is backtrack line search, a big alpha0 means we are initially far from the optimal alpha, thus taking more steps to get there.



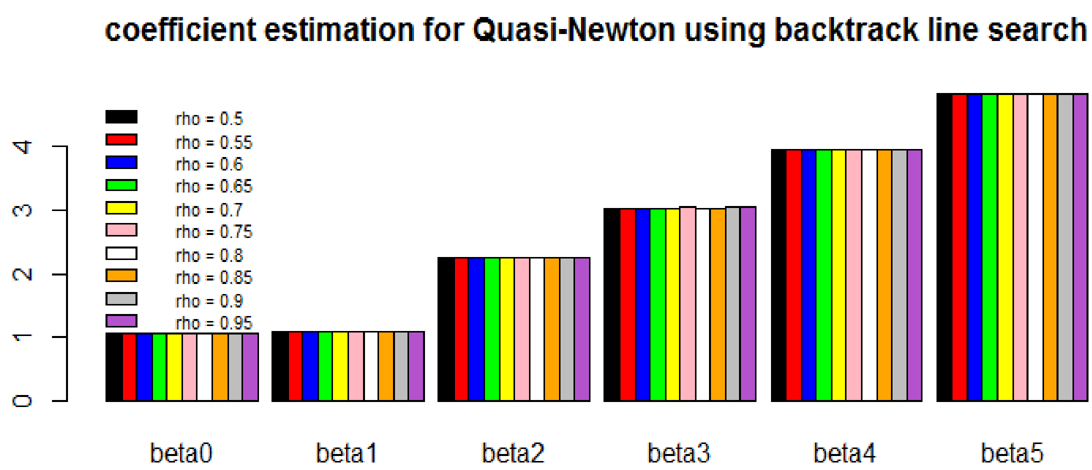

convergence of GD using backtrack line search w.r.t initial alpha
loglik
0
100
200
300
400
500
600
0
50
100
150
200
250
300
alpha0 = 0.1
alpha0 = 0.2
alpha0 = 0.3
alpha0 = 0.4
alpha0 = 0.5
alpha0 = 0.6
alpha0 = 0.7
alpha0 = 0.8
alpha0 = 0.9
alpha0 = 1

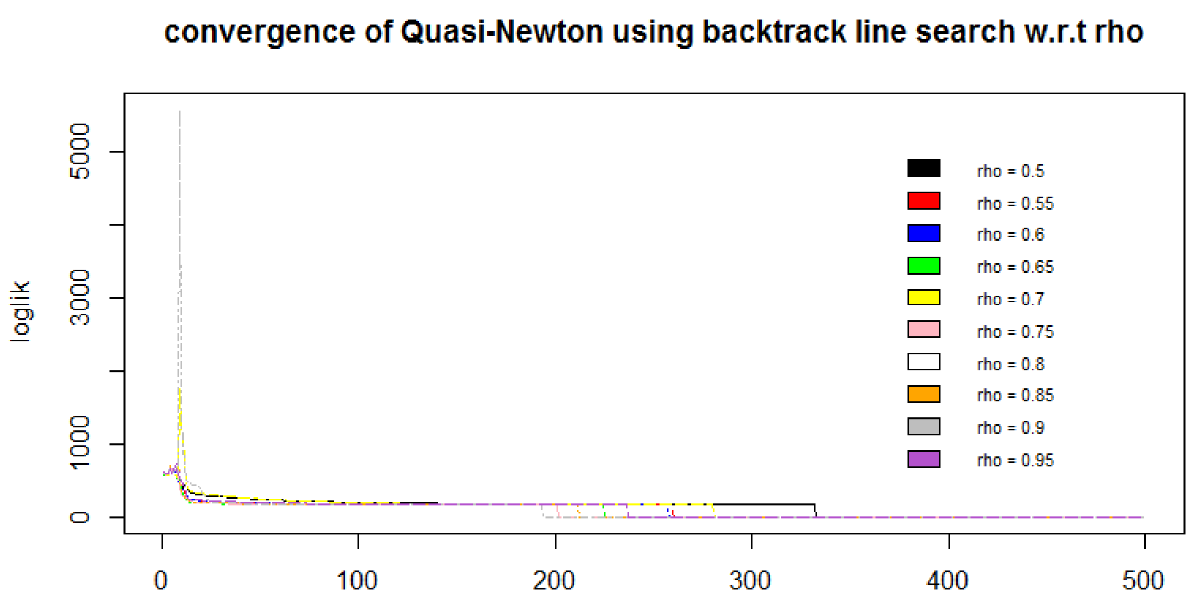
Quasi Newton method using BFGS updates and backtrack line search

We need to update step size, beta and approximate Hessian matrix in each iteration, and in that order.

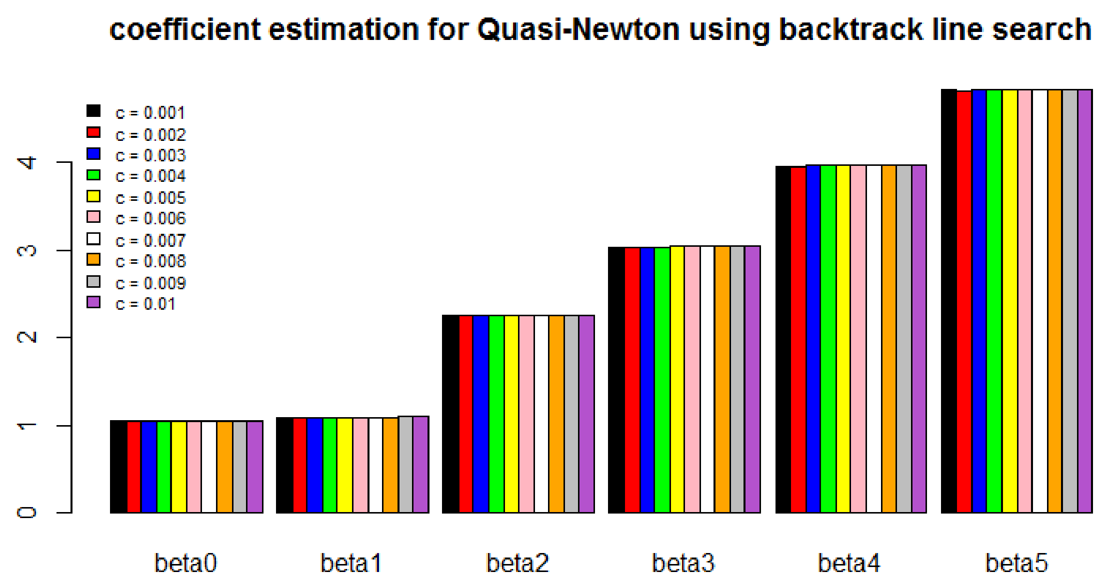
Performance with respect to rho (c = 0.001, alpha0 = 1)

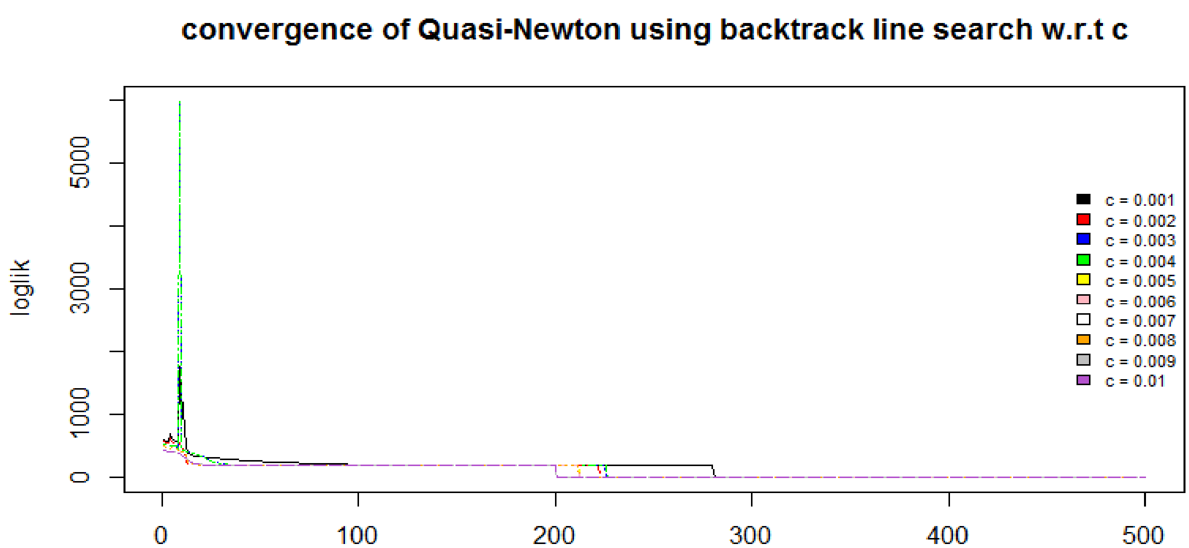
Estimation problem seems fixed when using Quasi-Newton.





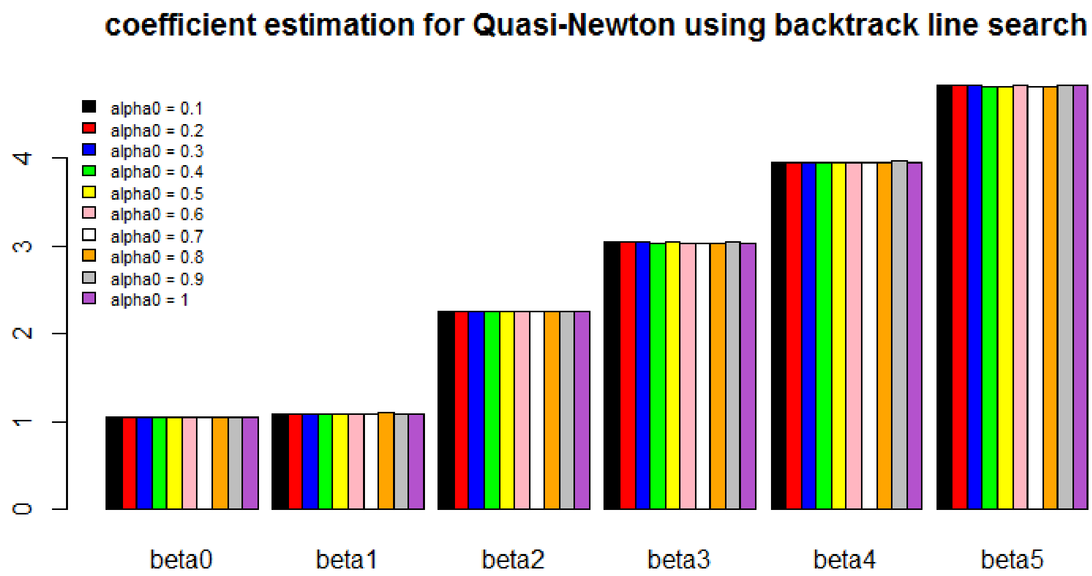
Performance with respect to c (rho = 0.7, alpha0 = 1)

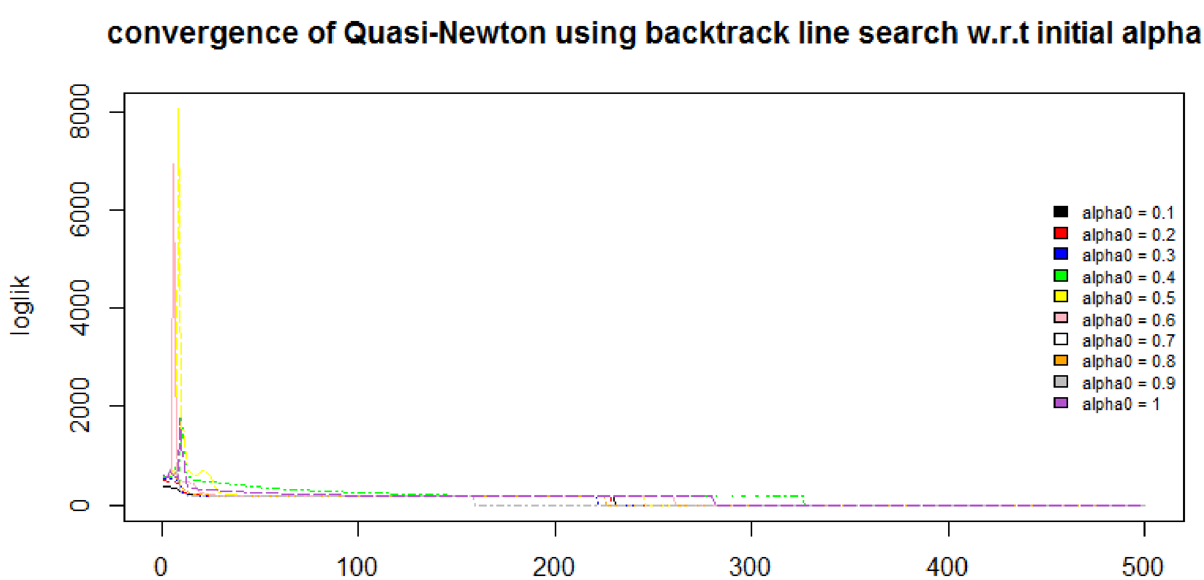




Performance with respect to initial alpha (rho = 0.7, c = 0.001)

Again, the estimation problem seems fixed under Quasi Newton method.





GD v.s. GD using backtrack line search v.s. Quasi-Newton using backtrack line search

Parameter setting:

GD: alpha = 0.01

GD using backtrack line search and Quasi-Newton using backtrack line search

Alpha0 = 1, c = 0.001, rho = 0.7

Test convergence based on log likelihood value. Threshold is set at 1e-5.

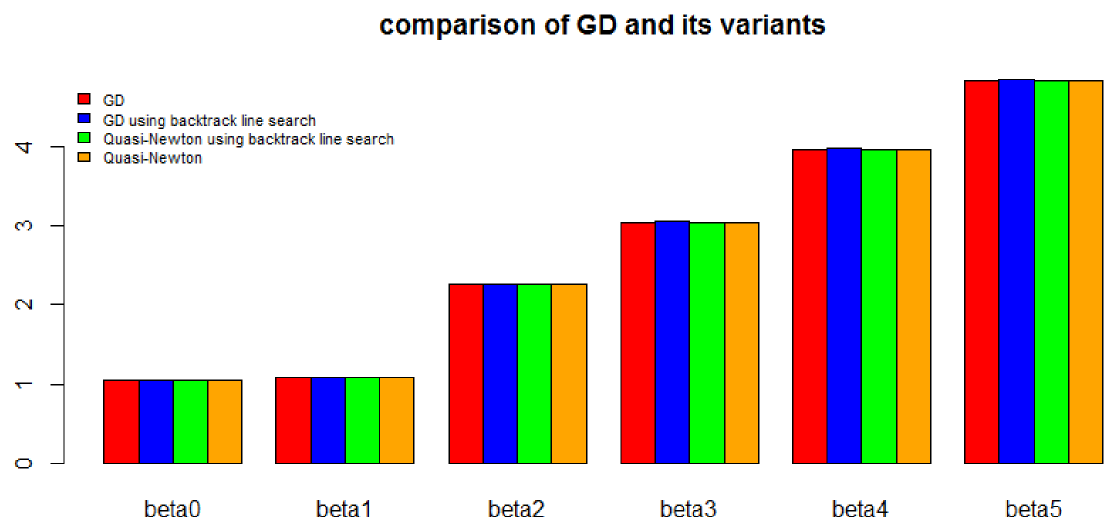
Takeaway. All algorithms return satisfactory estimates. Incorporating backtrack line search into GD takes fewer iterations to converge. But, unexpectedly, Quasi-Newton takes more iterations to converge than GD, no matter it includes backtrack line search or not. The reason might be attributed to the selection of our parameters. In terms of the time each algorithm runs, GD is fastest with mean milliseconds of 70.32, and Quasi-Newton using backtrack line search is slowest with mean milliseconds of 1024.66. So, backtrack line search is time consuming, but gives better step size and converges in fewer iterations.

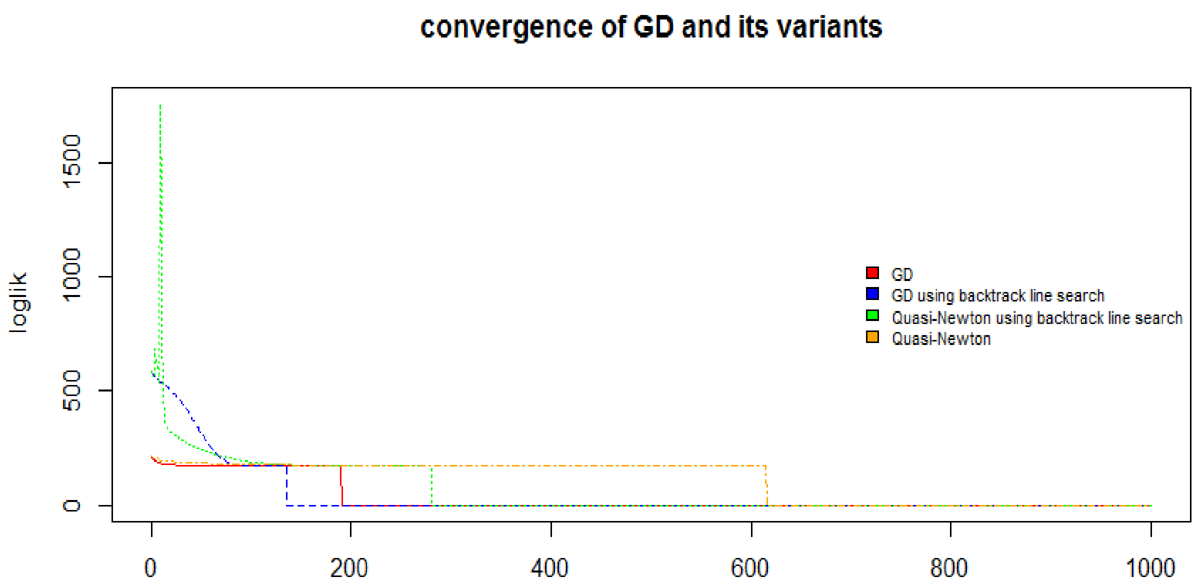
```
Unit: milliseconds
     expr      min       lq     mean   median       uq      max neval
 GD(X, y) 65.81369 67.54288 70.32293 68.33913 69.32054 122.2452   100

Unit: milliseconds
          expr      min       lq   mean  median       uq      max neval
 qNewton(X, y) 359.8116 366.5836 377.04 369.327 379.7995 468.2446   100

Unit: milliseconds
      expr      min       lq     mean   median       uq      max neval
 BGD(X, y) 372.2502 375.6868 379.2561 377.5591 379.9752 428.5745   100

Unit: milliseconds
             expr      min      lq     mean   median       uq      max neval
 qNewton_bb(X, y) 976.5969 984.299 1024.661 1025.465 1049.769 1122.962   100
```




convergence of GD and its variants
loglik
0
500
1000
1500
0
200
400
600
800
1000
GD
GD using backtrack line search
Quasi-Newton using backtrack line search
Quasi-Newton