SQUAREM: Accelerating the Convergence of EM, MM and Other Fixed-Point Algorithms

Ravi Varadhan and Yu Du

1 Overview of SQUAREM

"SQUAREM" is a package intended for accelerating slowly-convergent contraction mappings. It can be used for accelerating the convergence of slow, linearly convergent contraction mappings such as the EM (expectation-maximization) algorithm, MM (majorize and minimize) algorithm, and other nonlinear fixed-point iterations such as the power method for finding the dominant eigenvector. It uses a novel approach callled squared extrapolation method (SQUAREM) that was proposed in Varadhan and Roland (Scandinavian Journal of Statistics, 35: 335-353), and also in Roland, Vardhan, and Frangakis (Numerical Algorithms, 44: 159-172).

The functions in this package are made available with:

```
> library("SQUAREM")
```

You can look at the basic information on the package, including all the available functions with

```
> help(package = SQUAREM)
```

The package *setRNG* is not necessary, but if you want to exactly reproduce the examples in this guide then do this:

```
> require("setRNG")
> setRNG(list(kind = "Wichmann-Hill", normal.kind = "Box-Muller", seed = 123))
```

after which the example need to be run in the order here (or at least the parts that generate random numbers). For some examples the RNG is reset again so they can be reproduced more easily.

2 How to accelerate convergence of a fixed-point iteration with SQUAREM?

2.1 Accelerating EM algorithm: Binary Poisson Mixture Maximum-Likelihood Estimation

Now, we show an example demonstrating the ability of SQUAREM to dramatically speed-up the convergence of the EM algorithm for a binary Poisson mixture estimation. We use the example from Hasselblad (J of Amer Stat Assoc 1969)

```
> poissmix.dat <- data.frame(death = 0:9,
+                            freq = c(162, 267, 271, 185,
+                                     111, 61, 27, 8, 3, 1))
```

Generate a random initial guess for 3 parameters

```
> y <- poissmix.dat$freq
> tol <- 1.e-08
> setRNG(list(kind = "Wichmann-Hill", normal.kind = "Box-Muller", seed = 123))
> p0 <- c(runif(1),runif(2, 0, 6))
```

The fixed point mapping giving a single E and M step of the EM algorithm

```
> poissmix.em <- function(p, y) {
+         pnew <- rep(NA, 3)
+         i <- 0:(length(y) - 1)
+         zi <- p[1] * exp(-p[2]) * p[2]^i /
+                 (p[1]*exp(-p[2])*p[2]^i + (1 - p[1]) * exp(-p[3]) * p[3]^i)
+         pnew[1] <- sum(y * zi)/sum(y)
+         pnew[2] <- sum(y * i * zi)/sum(y * zi)
+         pnew[3] <- sum(y * i * (1-zi))/sum(y * (1-zi))
+         p <- pnew
+         return(pnew)
+ }
```

Objective function whose local minimum is a fixed point. Here it is the negative log-likelihood of binary poisson mixture.

```
> poissmix.loglik <- function(p, y) {
+         i <- 0:(length(y) - 1)
+         loglik <- y * log(p[1] * exp(-p[2]) * p[2]^i/exp(lgamma(i + 1)) +
+                           (1 - p[1]) * exp(-p[3]) * p[3]^i/exp(lgamma(i + 1)))
+         return (-sum(loglik))
+ }
```

EM algorithm

```
> pf1 <- fpiter(p = p0, y = y, fixptfn = poissmix.em, objfn = poissmix.loglik,
+               control = list(tol = tol))
> pf1

$par
[1] 0.6401136 2.6634056 1.2560968

$value.objfn
[1] 1989.946

$fpevals
[1] 2909

$objfevals
[1] 0

$convergence
[1] TRUE
```

Note the slow convergence of EM, as it uses more than 2900 iterations to converge. Now, let us speed up the convergence with SQUAREM:

```
> pf2 <- squarem(p = p0, y = y, fixptfn = poissmix.em, objfn = poissmix.loglik,
+                control = list(tol = tol))
> pf2

$par
[1] 0.6401146 2.6634044 1.2560951

$value.objfn
[1] 1989.946

$iter
[1] 25

$fpevals
[1] 72

$objfevals
[1] 25

$convergence
[1] TRUE
```

Note the dramatically faster convergence, i.e. SQUAREM uses only 72 fixed-point evaluations to achieve convergence. This is a speed up of a factor of 40.

We can also run SQUAREM without specifying an objective function, i.e. the negative log-likelihood. *This is usually the most efficient way to use SQUAREM.*

```
> pf3 <- squarem(p = p0, y = y, fixptfn = poissmix.em,
+                control = list(tol = tol))
> pf3


$par
[1] 0.6401146 2.6634044 1.2560951

$value.objfn
[1] NA

$iter
[1] 25

$fpevals
[1] 72

$objfevals
[1] 0

$convergence
[1] TRUE
```

2.2 Accelerating the Power Method for Finding the Dominant Eigenpair

The power method is a nonlinear fixed-point iteration for determining the dominant eigenpair, i.e. the largest eigenvalue (in terms of absolute magnitude) and the corresponding eigenvector, of a square matrix A. The iteration can be programmed in R as:

```
> power.method <- function(x, A) {
+ # Defines one iteration of the power method
+ # x = starting guess for dominant eigenvector
+ # A = a square matrix
+         ax <- as.numeric(A %*% x)
+         f <- ax / sqrt(as.numeric(crossprod(ax)))
+         f
+ }
```

We illustrate this for finding the dominant eigenvector of the Bodewig matrix which is a famous matrix for which power method has trouble converging. See, for example, Sidi, Ford, and Smith (SIAM Review, 1988). Here there are two eigenvalues that are equally dominant, but have opposite signs! Sometimes the power method finds the eigenvector corresponding to the large positive eigenvalue, but other times it finds the eigenvector corresponding to the large negative eigenvalue

```
> b <- c(2, 1, 3, 4, 1, -3, 1, 5, 3, 1, 6, -2, 4, 5, -2, -1)
> bodewig.mat <- matrix(b,4,4)
> eigen(bodewig.mat)

$values
[1]  7.932905  5.668864 -1.573191 -8.028578

$vectors
           [,1]       [,2]       [,3]       [,4]
[1,] -0.5601445  0.3787027  0.6880479  0.2634624
[2,] -0.2116328  0.3624190 -0.6241229  0.6590407
[3,] -0.7767083 -0.5379352 -0.2598009 -0.1996335
[4,] -0.1953816  0.6601988 -0.2637503 -0.6755734
```

Now, let us look at power method and its acceleration using various SQUAREM schemes:

```
> p0 <- rnorm(4)
> # Standard power method iteration
> ans1 <- fpiter(p0, fixptfn = power.method, A = bodewig.mat)
```

```
> # re-scaling the eigenvector so that it has unit length
> ans1$par <- ans1$par / sqrt(sum(ans1$par^2))
> # dominant eigenvector
> ans1


$par
[1]  0.2634624  0.6590407 -0.1996335 -0.6755734

$value.objfn
[1] NA

$fpevals
[1] 5000

$objfevals
[1] 0

$convergence
[1] FALSE

> # dominant eigenvalue
> c(t(ans1$par) %*% bodewig.mat %*% ans1$par) / c(crossprod(ans1$par))


[1] -8.028578
```

Now, we try first-order SQUAREM with default settings:

```
> ans2 <- squarem(p0, fixptfn = power.method, A = bodewig.mat)
> ans2


$par
[1] -0.2634624 -0.6590407  0.1996335  0.6755734

$value.objfn
[1] NA

$iter
[1] 751

$fpevals
[1] 1500

$objfevals
```

```
[1] 0

$convergence
[1] FALSE
```

```
> ans2$par <- ans2$par / sqrt(sum(ans2$par^2))
> c(t(ans2$par) %*% bodewig.mat %*% ans2$par) / c(crossprod(ans2$par))
```

```
[1] -8.028578
```

The convergence is still slow, but it converges to the dominant eigenvector. Now, we try with a minimum steplength that is smaller than 1.

```
> ans3 <- squarem(p0, fixptfn = power.method, A = bodewig.mat,
+                 control = list(step.min0 = 0.5))
> ans3

$par
[1] 0.5601445 0.2116328 0.7767083 0.1953816

$value.objfn
[1] NA

$iter
[1] 10

$fpevals
[1] 27

$objfevals
[1] 0

$convergence
[1] TRUE
```

```
> ans3$par <- ans3$par / sqrt(sum(ans3$par^2))
> # eigenvalue
> c(t(ans3$par) %*% bodewig.mat %*% ans3$par) / c(crossprod(ans3$par))
```

```
[1] 7.932905
```

The convergence is dramatically faster now, but it converges to the second dominant eigenvector. We try again with a higher-order SQUAREM scheme.

```
> # Third-order SQUAREM
> ans4 <- squarem(p0, fixptfn = power.method, A = bodewig.mat,
+                 control = list(K = 3, method = "rre"))
> ans4


$par
[1] 0.5601445 0.2116328 0.7767083 0.1953816

$value.objfn
[1] NA

$iter
[1] 5

$fpevals
[1] 29

$objfevals
[1] 0

$convergence
[1] TRUE


> ans4$par <- ans4$par / sqrt(sum(ans4$par^2))
> # eigenvalue
> c(t(ans4$par) %*% bodewig.mat %*% ans4$par) / c(crossprod(ans4$par))


[1] 7.932905
```

Once again we obtain the second dominant eigenvector.

2.3 Factor Analysis

Factor analysis is a statistical modeling approach that aims to explain the variability among observed variables in terms of a smaller set of unobserved factors. Factor analysis is widely applied in areas where observed variables may be conceptualized as manifesting from some unobserved latent factors, such as psychometrics, behavioral sciences, social sciences, and marketing. The latent factors can be regarded as missing data in a multivariate normal model and the EM algorithm, therefore, becomes a natural way to compute the maximum likelihood estimates. We will illustrate the dramatic acceleration of EM by Squarem and also compare with ECME (Liu and Rubin 1998) using a real data example from JoresKog (1967).

The data consists of 9 variables, 4 factors, and 2 patterns of a priori zeroes for the loadings such that one a priori zero loadings on factor 4 for variables 1 through 4, and a different a priori zero loadings on factor 3 for variables 5-9. There is otherwise no restrictions. The sample covariance matrix C_{yy} is given below:

$$C_{yy} = \begin{pmatrix} 1.0 & 0.554 & 0.227 & 0.189 & 0.461 & 0.506 & 0.408 & 0.280 & 0.241 \\ & 1.0 & 0.296 & 0.219 & 0.479 & 0.530 & 0.425 & 0.311 & 0.311 \\ & & 1.0 & 0.769 & 0.237 & 0.243 & 0.304 & 0.718 & 0.730 \\ & & & 1.0 & 0.212 & 0.226 & 0.291 & 0.681 & 0.661 \\ & & & & 1.0 & 0.520 & 0.514 & 0.313 & 0.245 \\ & & & & & 1.0 & 0.473 & 0.348 & 0.290 \\ & & & & & & 1.0 & 0.374 & 0.306 \\ & & & & & & & 1.0 & 0.672 \\ & & & & & & & & 1.0 \end{pmatrix}.$$

We use the starting values of β and τ^2 as in Liu and Rubin (1998), where

$$\beta'^{\text{start}} = \begin{pmatrix} 0.5954912 & -0.4893347 & -0.3848925 & 0.0000000 \\ 0.6449102 & -0.4408213 & -0.3555598 & 0.0000000 \\ 0.7630006 & 0.5053083 & -0.0535340 & 0.0000000 \\ 0.7163828 & 0.5258722 & 0.0219100 & 0.0000000 \\ 0.6175647 & -0.4714808 & 0.0000000 & 0.1931459 \\ 0.6464100 & -0.4628659 & 0.0000000 & 0.4606456 \\ 0.6452737 & -0.3260013 & 0.0000000 & -0.3622682 \\ 0.7868222 & 0.3690580 & 0.0000000 & 0.0630371 \\ 0.7482302 & 0.4326963 & 0.0000000 & 0.0431256 \end{pmatrix},$$

and $\tau_j^{2\text{start}} = 10^{-8}$ for $j = 1, 2, \ldots, 9$.

Here is the negative log likelihood, given by the function **factor.loglik()**:

```
> factor.loglik <- function(param, cyy){
+         ###extract beta matrix and tau2 from param
```

```
+         beta.vec <- param[1:36]
+         beta.mat <- matrix(beta.vec, 4, 9)
+         tau2 <- param[37:45]
+         tau2.mat <- diag(tau2)
+
+         Sig <- tau2.mat + t(beta.mat) %*% beta.mat
+         ##suppose n=145 since this does not impact the parameter estimation
+         loglik <- -145/2 * log(det(Sig)) - 145/2 * sum(diag(solve(Sig, cyy)))
+         return(-loglik)
+         ###the negative log-likelihood is returned
+ }
```

The fixed point mapping giving a single E and M step of the EM algorithm.

```
> factor.em <- function(param, cyy){
+         param.new <- rep(NA, 45)
+
+         ###extract beta matrix and tau2 from param
+         beta.vec <- param[1:36]
+         beta.mat <- matrix(beta.vec, 4, 9)
+         tau2 <- param[37:45]
+         tau2.mat <- diag(tau2)
+
+         ###compute delta/Delta
+         inv.quantity <- solve(tau2.mat + t(beta.mat) %*% beta.mat)
+         small.delta <- inv.quantity %*% t(beta.mat)
+         big.delta <- diag(4) - beta.mat %*% inv.quantity %*% t(beta.mat)
+
+         cyy.inverse <- t(small.delta) %*% cyy %*% small.delta + big.delta
+         cyy.mat <- t(small.delta) %*% cyy
+
+         ###update betas and taus
+         beta.new <- matrix(0, 4, 9)
+         beta.p1 <- solve(cyy.inverse[1:3, 1:3]) %*% cyy.mat[1:3, 1:4]
+         beta.p2 <- solve(cyy.inverse[c(1,2,4), c(1,2,4)]) %*%
+                    cyy.mat[c(1,2,4), 5:9]
+         beta.new[1:3, 1:4] <- beta.p1
+         beta.new[c(1,2,4), 5:9] <- beta.p2
+
+         tau.p1 <- diag(cyy)[1:4] - diag(t(cyy.mat[1:3, 1:4]) %*%
+                   solve(cyy.inverse[1:3, 1:3]) %*% cyy.mat[1:3, 1:4])
+         tau.p2 <- diag(cyy)[5:9] - diag(t(cyy.mat[c(1,2,4), 5:9]) %*%
+                   solve(cyy.inverse[c(1,2,4), c(1,2,4)]) %*%
+                   cyy.mat[c(1,2,4), 5:9])
```

```
+         tau.new <- c(tau.p1, tau.p2)
+
+         param.new <- c(as.numeric(beta.new), tau.new)
+         param <- param.new
+         return(param.new)
+ }
```

In order to compare with ECME algorithm as implemented by Liu and Rubin (1998), we also write the function **factor.ecme()** to do one ECME iteration. The only difference from EM algorithm is that for M-Step, after we update the loading matrix β, we find τ^2 that maximizes the actual constrained likelihood of observed data matrix Y given β using Newton-Raphson.

```
> factor.ecme <- function(param, cyy){
+         n <- 145
+         param.new <- rep(NA, 45)
+
+         ###extract beta matrix and tau2 from param
+         beta.vec <- param[1:36]
+         beta.mat <- matrix(beta.vec, 4, 9)
+         tau2 <- param[37:45]
+         tau2.mat <- diag(tau2)
+
+         ###compute delta/Delta
+         inv.quantity <- solve(tau2.mat + t(beta.mat) %*% beta.mat)
+         small.delta <- inv.quantity %*% t(beta.mat)
+         big.delta <- diag(4) - beta.mat %*% inv.quantity %*% t(beta.mat)
+
+         cyy.inverse <- t(small.delta) %*% cyy %*% small.delta + big.delta
+         cyy.mat <- t(small.delta) %*% cyy
+
+         ###update betas
+         beta.new <- matrix(0, 4, 9)
+         beta.p1 <- solve(cyy.inverse[1:3, 1:3]) %*% cyy.mat[1:3, 1:4]
+         beta.p2 <- solve(cyy.inverse[c(1,2,4), c(1,2,4)]) %*%
+                    cyy.mat[c(1,2,4), 5:9]
+         beta.new[1:3, 1:4] <- beta.p1
+         beta.new[c(1,2,4), 5:9] <- beta.p2
+
+         ###update taus given betas
+         A <- solve(tau2.mat + t(beta.new) %*% beta.new)
+         sum.B <- A %*% (n * cyy) %*% A
+         gradient <- - tau2/2 * (diag(n*A) - diag(sum.B))
+         hessian <- (0.5 * (tau2 %*% t(tau2))) * (A * (n * A - 2 * sum.B))
```

```
+         diag(hessian) <- diag(hessian) + gradient
+         U <- log(tau2)
+         U <- U - solve(hessian, gradient)  # Newton step
+
+         tau.new <- exp(U)
+         param.new <- c(as.numeric(beta.new), tau.new)
+         param <- param.new
+         return(param.new)
+ }
```

Now we can use SQUAREM to compute the MLE by EM, Squared EM (Squarem), ECME, and Squared ECME algorithms. Tolerance is set to be 10^{-8} across all algorithms.

- Basic EM

  In order to perform the basic EM algorithm, we use function **fpiter()** in SQUAREM Package. The arguments consist of a starting value, function **factor.em()** that encodes one EM update, the negative log likelihood function **factor.loglik()**, other variables as needed by these functions, and a control list to specify changes to default values. The starting value for β and τ^2 comes from Liu and Rubin (1998).

```
> library(SQUAREM)
> system.time(f1 <- fpiter(par = param.start, cyy = cyy,
+                          fixptfn = factor.em,
+                          objfn = factor.loglik,
+                          control = list(tol=10^(-8),
+                          maxiter = 20000)))

   user  system elapsed
  2.541   0.025   2.566

> f1$fpevals

[1] 14659
```

  It takes 14659 iterations to converge for basic EM algorithm.

- ECME

  We replace function **factor.em()** by **factor.ecme()** that implements one ECME update thus to implement ECME algorithm.

```
> system.time(f2 <- fpiter(par = param.start, cyy = cyy,
+                          fixptfn = factor.ecme, objfn = factor.loglik,
+                          control = list(tol=10^(-8), maxiter = 20000)))
```

```
   user  system elapsed
  1.226   0.030   1.257
```

```
> f2$fpevals
```

```
[1] 6408
```

It takes 6409 iterations of ECME update to converge, less than half of what basic EM needs.

- Squared EM (Squarem)

  Next, we use function **squarem()** in SQUAREM Pakcage to apply Squarem algorithm. The components of arguments are the same as function **fpiter()** except a few control parameters particularly set for Squarem.

```
> system.time(f3 <- squarem(par = param.start, cyy = cyy,
+                           fixptfn = factor.em,
+                           objfn = factor.loglik,
+                           control = list(tol = 10^(-8))))
```

```
   user  system elapsed
  0.198   0.003   0.201
```

```
> f3$fpevals
```

```
[1] 876
```

It only takes 876 iterations of EM updates to converge, which is faster by a factor of 17 and 7 in terms of the number of EM steps when compared to the basic EM and ECME, respectively.

- Squared ECME

  Squarem can even accelerate ECME, which is already a faster version of the basic EM algorithm.

```
> system.time(f4 <- squarem(par = param.start, cyy = cyy,
+                           fixptfn = factor.ecme,
+                           objfn = factor.loglik,
+                           control = list(tol = 10^(-8))))
```

```
   user  system elapsed
  0.094   0.001   0.096
```

```
> f4$fpevals
```

```
[1] 400
```

The squared ECME converges in only 400 iterations compared to 6400 iterations for ECME, the fastest among these four algorithms.

2.4 Interval Censoring

Interval censoring is a common phenomenon in survival analysis, where we do not observe the precise time of an event for each individual, but we only know the time interval during which the individual's event occurs. Following the notations in Gentleman and Geyer (1994), we assume that survival time, X, also known as failure time, come from a distribution F. Each individual i goes through a sequence of inspection times $t_{i,1}, t_{i,2}, \ldots$. The survival time x_i for individual i is not observed, however, the last inspection time prior to x_i and the first inspection time after are recorded. An example of interval censored data is displayed in Table 1.

Table 1: The Example of Interval Censored Data(Unit: Year).

	Last Inspection Time prior to x_i	First Inspection Time after x_i
Individual 1	1	3
Individual 2	2	6
$\vdots$	$\vdots$	$\vdots$
Individual n	3	4

Therefore, data consists of time intervals $I_i = (L_i, R_i)$ for each individual $i, i = 1, 2, \ldots, n$ and the event for individual i is known to happen during that interval. Let $\{s_j\}_{j=0}^m$ be the unique ordered times of $\{0, \{L_i\}_{i=1}^n, \{R_i\}_{i=1}^n\}$, $\alpha_{ij}, i = 1, 2, \ldots, n, j = 1, 2, \ldots, m$, the ij cell of an α matrix, be such that

$$\alpha_{ij} = \begin{cases} 1 & \text{if } (s_{j-1}, s_j) \subseteq I_i, \text{ the event for individual } i \text{ can occur in } (s_{j-1}, s_j) \\ 0 & \text{otherwise} \end{cases}$$

and $p_j = F(s_j-) - F(s_{j-1})$, $p = (p_1, p_2, \ldots, p_m)'$. The log likelihood of data is therefore

$$ll(p) = \sum_{i=1}^{n} \log\left(\sum_{j=1}^{m} \alpha_{ij} p_j\right).$$

The negative log likelihood is coded in function **loglik()**.

```
> loglik <- function(pvec, A){
+         - sum(log(c(A %*% pvec)))
+ }
> ##A is the alpha matrix and pvec is the vector of probabilities p.
> ##returns the negative log likelihood
```

One EM update is coded in function **intEM()**.

```
> intEM <- function(pvec, A){
+         tA <- t(A)
+         Ap <- pvec*tA
+         pnew <- colMeans(t(Ap)/colSums(Ap))
+         pnew * (pnew > 0)
+ }
> ##tA is the transpose of alpha matrix A
```

Now we demonstrate the acceleration of EM by Squarem using the real data example from Finkelstein and Wolfe (1985), which gives the interval when cosmetic deterioration occurred in 46 individuals with early breast cancer under radiotherapy. Table 2 shows censored interval for each individual.

Table 2: Censored Intervals when cosmetic deterioration occurred

(45,Inf]	(6,10]	(0,7]	(46,Inf]	(7,16]	(17,Inf]
(7,14]	(37,44]	(0,8]	(4,11]	(15,Inf]	(11,15]
(22,Inf]	(46,Inf]	(46,Inf]	(25,37]	(46, Inf]	(26, 40]
(46, Inf]	(27,34]	(36,44]	(46, Inf]	(36, 48]	(37, Inf]
(40, Inf]	(17,25]	(46, Inf]	(11,18]	(38, Inf]	(5, 12]
(37, Inf]	(0,5]	(18, Inf]	(24, Inf]	(36, Inf]	(5, 11]
(19, 35]	(17,25]	(24, Inf]	(32, Inf]	(33, Inf]	(19,26]
(37,Inf]	(34,Inf]	(36, Inf]	(46,Inf]		

We use function **Aintmap()** in R Package interval to produce intervals (s_{j-1}, s_j), $j = 1, 2, \ldots, m$.

```
> library(interval)
```

```
> A <- Aintmap(dat[,1], dat[,2])
> m <- ncol(A)
> ##starting values
> pvec <- rep(1/m, length = m)
> ##EM
> system.time(ans1 <- fpiter(par = pvec, fixptfn = intEM,
+                            objfn = loglik, A = A,
+                            control = list(tol = 1e-8)))
```

```
   user  system elapsed
  0.006   0.001   0.008
```

```
> ans1
```

```
$par
       (4,5]        (6,7]        (7,8]      (11,12]      (15,16]      (17,18]
4.634677e-02 3.336337e-02 8.866737e-02 7.075292e-02 1.505666e-72 2.936685e-19
     (24,25]      (25,26]      (33,34]      (34,35]      (36,37]      (38,40]
9.264584e-02 9.471938e-24 8.178576e-02 1.538862e-26 1.879768e-08 1.208797e-01
     (40,44]      (46,48]
1.095534e-07 4.655581e-01

$value.objfn
[1] 58.06002

$fpevals
[1] 216

$objfevals
[1] 0

$convergence
[1] TRUE

> ##Squarem
> system.time(ans2 <- squarem(par = pvec, fixptfn = intEM,
+                             objfn = loglik, A = A,
+                             control = list(tol = 1e-8)))

   user  system elapsed
  0.002   0.000   0.003

> ans2

$par
       (4,5]        (6,7]        (7,8]      (11,12]      (15,16]      (17,18]
4.634677e-02 3.336338e-02 8.866737e-02 7.075292e-02 4.593510e-12 2.804527e-09
     (24,25]      (25,26]      (33,34]      (34,35]      (36,37]      (38,40]
9.264585e-02 1.795427e-10 8.178576e-02 1.462811e-11 2.024202e-08 1.208798e-01
     (40,44]      (46,48]
6.821933e-08 4.655581e-01

$value.objfn
[1] 58.06002

$iter
[1] 14
```

```
$fpevals
[1] 40

$objfevals
[1] 14

$convergence
[1] TRUE

>
```

Even with this small sample size, the Squarem still improves on EM by a factor of 5 in terms of the number of EM evaluations, 40 iterations compared to 216. As sample size expands, the margin of the advantages for Squarem over EM will get much larger.

2.5 MM Algorithm - Logistic Regression Maximum Likelihood Estimation

In this section, we discuss a quadratic majorization algorithm (an MM algorithm) for computing the maximum likelihood estimates of logistic regression coefficients. Minorize and maximize or equivalently, majorize and minimize (MM) algorithms typically exhibit slow linear convergence just like the EM algorithms. We show that Squarem can provide significant acceleration of MM algorithms.

Suppose we want to minimize function f over $X \subseteq \mathbb{R}^n$. We construct a majorization function g on $X \times X$ such that

$$f(x) \leq g(x, x^{(k)}) \quad \forall x, x^{(k)} \in X,$$

$$f(x^{(k)}) = g(x^{(k)}, x^{(k)}) \quad \forall x^{(k)} \in X,$$

where k denotes the k^{th} iteration, $k = 0, 1, \ldots$. Therefore, instead of minimizing f, we minimize g such that

$$x^{(k+1)} = \underset{x \in X}{\operatorname{argmin}}\, g(x, x^{(k)}).$$

We repeat the updates of x until convergence and this completes the majorization algorithm. Note that in the EM algorithm, the $Q(\theta; \theta_k)$ function plays the role of the minorizing function.

Taylor's theorem often leads to quadratic majorization algorithms (Bohning and Lindsay 1988) where the majorization function g is quadratic. By Taylor's theorem, expand $f(x)$ at $x^{(k)}$,

$$f(x) = f(x^{(k)}) + (x - x^{(k)})'\partial f(x^{(k)}) + \frac{1}{2}(x - x^{(k)})'\partial^2 f(\xi)(x - x^{(k)})$$

where ξ is on the line between x and $x^{(k)}$. The majorization function g is constructed by constructing a matrix B such that $B - \partial^2 f(\xi)$ is always positive semi-definite regardless of ξ. So,

$$g(x, x^{(k)}) = f(x^{(k)}) + (x - x^{(k)})'\partial f(x^{(k)}) + \frac{1}{2}(x - x^{(k)})'B(x - x^{(k)})$$

is a majorization function for f. Let us define a clever variable $z^{(k)}$ such that $z^{(k)} = x^{(k)} - B^{-1}\partial f(x^{(k)})$ and majorization function g is equivalent to the following:

$$g(x, x^{(k)}) = f(x^{(k)}) + \frac{1}{2}(x - z^{(k)})'B(x - z^{(k)}) - \frac{1}{2}\partial f(x^{(k)})'B^{-1}\partial f(x^{(k)}).$$

At the k^{th} iteration, to minimize $g(x, x^{(k)})$ over $x \in X$ is simply to minimize $(x - z^{(k)})'B(x - z^{(k)})$, thus the majorization algorithm becomes:

$$x^{(k+1)} = x^{(k)} - B^{-1}\partial f(x^{(k)}).$$

Therefore, in order to implement quadratic majorization algorithm, we need to construct the matrix B and compute the gradient of function f.

Let us consider logistic regression maximum likelihood estimation. Suppose we have an $n \times p$ design matrix X where there are n subjects and p predictors. Let y_i be the number

of successes for subject i ,$i = 1, 2, \ldots, n$ given the overall number of experiments, N_i. We use β to denote the regression coefficients. The goal is to derive the maximum likelihood estimates of β.

The negative log likelihood of data is:

$$\begin{aligned} f(\beta) &= -\log\left(\prod_i P(y_i)\right) \\ &\propto -\log\left(\prod_i p_i(\beta)^{y_i}(1-p_i(\beta))^{(N_i-y_i)}\right) \\ &= -\sum_i y_i \log p_i(\beta) - \sum_i (N_i - y_i)\log\left(1-p_i(\beta)\right) \\ &= -\sum_i y_i x_i'\beta - \sum_i N_i \log\left(1-p_i(\beta)\right), \end{aligned}$$

where

$$p_i(\beta) = \frac{1}{1+\exp\left(-x_i'\beta\right)}.$$

The gradient of $f(\beta)$ is such that:

$$\partial f(\beta) = \sum_i (N_i p_i(\beta) - y_i)x_i = X'u(\beta),$$

where the i^{th} element of $u(\beta)$ is $N_i p_i(\beta) - y_i$, while the second derivative of $f(\beta)$ is:

$$\partial^2 f(\beta) = \sum_i (N_i p_i(\beta)(1-p_i(\beta)))x_i x_i' = X'V(\beta)X,$$

where $V(\beta)$ is a diagonal matrix with the i^{th} diagonal element $N_i p_i(\beta)(1-p_i(\beta))$. Based on the fact that $p_i(\beta)(1-p_i(\beta)) \leq \frac{1}{4}$, the matrix B can be constructed such that $B = \frac{1}{4}X'NX$ where N is the diagonal matrix consisting of elements N_i. Thus the quadratic algorithm becomes:

$$\beta^{(k+1)} = \beta^{(k)} - 4(X'NX)^{-1}X'u(\beta).$$

Let us denote the above algorithm by uniform bound quadratic algorithm since $p_i(\beta)(1-p_i(\beta)) \leq \frac{1}{4}$ uniformly for any β and each subject i. Jaakkola and Jordan (2000) and Groenen, Giaquinto, and Kiers (2003) developed a non-uniform bound, $X'W(\beta)X$, where $W(\beta)$ is a diagonal matrix that consists of elements $w_i(\beta) = N_i\frac{2p_i(\beta)-1}{2x_i'\beta}$, $i = 1, 2, \ldots, n$. Thus the non-uniform bound quadratic algorithm becomes:

$$\beta^{(k+1)} = \beta^{(k)} - (X'W(\beta)X)^{-1}X'u(\beta).$$

We use the Cancer Remission data in Lee (1974). The outcome is a binary indicator of whether cancer remission occurred for the subject. Column 1 is the intercept and variables $X2$, $X3$, ..., $X7$ are results of six medical tests. The first five lines of data are as follows:

```
> head(ld, 5)
```

```
  X1  X2   X3   X4  X5    X6    X7 X8
1  1 0.8 0.83 0.66 1.9 1.100 0.996  1
2  1 0.9 0.36 0.32 1.4 0.740 0.992  1
3  1 0.8 0.88 0.70 0.8 0.176 0.982  0
4  1 1.0 0.87 0.87 0.7 1.053 0.986  0
5  1 0.9 0.75 0.68 1.3 0.519 0.980  1
```

The negative log likelihood function $f(\beta)$ is coded in **binom.loglike()**.

```
> binom.loglike <- function(par, Z, y){
+         zb <- c(Z %*% par)
+         pib <- 1 / (1 + exp(-zb))
+         return(as.numeric(-t(y) %*% (Z %*% par) - sum(log(1 - pib))))
+ }
```

The uniform bound quadratic majorization algorithm update and the non-uniform one are coded in function **qmub.update()**, **qmvb.update()** respectively.

```
> #######quadratic majorization uniform bound##########
> qmub.update <- function(par, Z, y){
+         Zmat <- solve(crossprod(Z)) %*% t(Z)
+         zb <- c(Z %*% par)
+         pib <- 1 / (1 + exp(-zb))
+         ub <-  pib - y
+         par <- par - 4 * c(Zmat %*% ub)
+         par
+ }
> ######quadratic majorization non uniform bound#######
> qmvb.update <- function(par, Z, y){
+         zb <- c(Z %*% par)
+         pib <- 1 / (1 + exp(-zb))
+         wmat <- diag((2 * pib - 1)/(2 * zb))
+         ub <-  pib - y
+         Zmat <- solve(t(Z) %*% wmat %*% Z) %*% t(Z)
+         par <- par - c(Zmat %*% ub)
+         par
+ }
```

Now let us apply these two quadratic majorization algorithms and their Squared versions to compare their performance. The tolerance used is 10^{-7} and the starting value is $\beta^{(0)} = (10, 10, \ldots, 10)'$.

```
> library(SQUAREM)
> Z <- as.matrix(ld[, 1:7])
> y <- ld[, 8]
> p0 <- rep(10, 7)
> ###uniform bound###
> system.time(ans1 <- fpiter(par = p0, fixptfn = qmub.update,
+                            objfn = binom.loglike,
+                            control = list(maxiter = 20000),
+                            Z = Z, y = y))

   user  system elapsed
  0.041   0.000   0.041

> ans1

$par
[1]  58.0384838  24.6615508  19.2935824 -19.6012695   3.8959635   0.1510923
[7] -87.4339059

$value.objfn
[1] 10.87533

$fpevals
[1] 1127

$objfevals
[1] 0

$convergence
[1] TRUE

> ###squared uniform bound###
> system.time(ans2 <- squarem(par = p0, fixptfn = qmub.update,
+                             objfn = binom.loglike,
+                             Z = Z, y = y))

   user  system elapsed
  0.008   0.000   0.007

> ans2

$par
[1]  58.0384863  24.6615466  19.2935777 -19.6012645   3.8959634   0.1510923
```

```
[7] -87.4339043

$value.objfn
[1] 10.87533

$iter
[1] 41

$fpevals
[1] 118

$objfevals
[1] 43

$convergence
[1] TRUE
```

```
> ###non-uniform bound###
> system.time(ans3 <- fpiter(par = p0, fixptfn = qmvb.update,
+                            objfn = binom.loglike,
+                            control = list(maxiter = 20000),
+                            Z = Z, y = y))
```

```
   user  system elapsed
  0.024   0.000   0.024
```

```
> ans3
```

```
$par
[1]  58.0384866  24.6615451  19.2935760 -19.6012627   3.8959634   0.1510923
[7] -87.4339030

$value.objfn
[1] 10.87533

$fpevals
[1] 442

$objfevals
[1] 0

$convergence
[1] TRUE
```

```
> ###squared non-uniform bound###
> system.time(ans4 <- squarem(par = p0, fixptfn = qmvb.update,
+                             objfn = binom.loglike,
+                             Z = Z, y = y))

   user  system elapsed
  0.006   0.000   0.007

> ans4

$par
[1]  58.0384868  24.6615443  19.2935751 -19.6012618   3.8959633   0.1510923
[7] -87.4339024

$value.objfn
[1] 10.87533

$iter
[1] 30

$fpevals
[1] 88

$objfevals
[1] 30

$convergence
[1] TRUE
```

All four algorithms converge to the same maximum likelihood estimates but Squarem improves on both uniform and non-uniform bound quadratic majorization algorithms in terms of the number of quadratic majorization updates and CPU running time (in seconds). For uniform bound, its Squared version converges around 6 times faster and saves the number of quadratic majorization updates by a factor of 10 (118 iterations vs 1127). The non-uniform bound quadratic majorization improves on the uniform bound one, but the Squared version of non-uniform bound quadratic majorization provides further acceleration. Compared to non-uniform bound, its Squared version shortens the computing time by a factor of 3 and cuts the number of quadratic majorization updates by a factor of 5 (88 iterations vs 442).