SNV Genotyping

GENOME 541: Cancer Genomics - Probabilistic Methods for Detecting Mutations in Cancer Genomes from Sequencing Data (WGS) University of Washington Summer 2020

Lavinia Carabet

8/28/2020

Implementation of a single-nucleotide variant (SNV) caller using a standard, single-sample binomial mixture model, an unsupervised and generative Bayesian statistical model.

Implementation of the expectation-maximization (EM) algorithm to predict the mutation status (`NotSNV` or `SNV`) for each genomic locus in input data.

The input data `alleleCounts.txt` contains a set of 1000 loci with reference and non-reference allele counts.

Overall, implementation of mixture model functions, implementation and execution of the EM algorithm to convergence, and lastly annotatation of the mutation status for each locus.

Steps:

0. Initial setup of the libraries and data
1. Implementation of the binomial likelihood model
2. Implementation of the functions for the EM algorithm
3. Implementation and execution of the full EM algorithm to learn the parameters and infer the genotypes.

0. Setup the libraries and input data

0.1. Load libraries

```
library(ggplot2) # for plotting in Section 3
```

0.2. Load the input data

The file containing allelic counts for T genomic loci. Each locus i contains the reference base (A, `refCount`) count (x_i) and non-reference/variant allele (B, `NrefCount`) base count. The read depth (N_i, `depth`) at each locus i is the sum of the counts for both bases $A + B$ at that locus.

```
counts <- read.delim("alleleCounts.txt", as.is = TRUE)
counts[1:2, ]
```

```
##   chr position refBase refCount NRefBase NrefCount depth
## 1   2    91822       C       42        N         2    44
## 2   2   202466       C       36        N        12    48
```

```
x <- counts$refCount
N <- counts$depth
K <- 3  # number of possible genotypes: AA (homozygous reference), AB (heterozygous variant), BB (homoz
```

0.3. Initialize model parameters and hyperparameters.

Initialize parameters and hyperparameters for genotypes, AA, AB, BB,

- `mu.init`, $\mu_{1:K}^{(0)}$ for the binomial model parameter (the probability of observing a reference base for the 3 genotypes)
- `pi.init`, $\pi_{1:K}^{(0)}$ for the genotype probability (the mixture weights/proportions that sum to 1)
- `alpha.hyper`, $\alpha_{1:K}$ for the Beta prior hyperparameter (μ can be sampled from a Beta distribution with α and β parameters and which is in the interval [0,1])
- `beta.hyper`, $\beta_{1:K}$ for the Beta prior hyperparameter
- `delta.hyper`, $\delta_{1:K}$ for the Dirichlet prior hyperparameter (mixing genotype proportions, π, follow a Dirichlet distribution with parameter δ)

```
mu.init <- c(0.99, 0.5, 0.01)
pi.init <- c(0.80, 0.15, 0.05)
alpha.hyper <- c(10, 5, 1)
beta.hyper <- c(1, 5, 10)
delta.hyper <- c(8, 2, 2)
```

1. Implement functions for the Binomial Mixture Model

1.1. Compute the observed binomial likelihood probabilities

1.1.1. Define a function, `compute.binom.lik` This function will compute the binomial probability for the input x_i at each locus i and (conditioned on) each genotype $k \in \{AA, AB, BB\}$. This function thus computes the observed likelihood probabilities for the 3-component mixture of binomials with 3 parameters, μ_k, one for each genotype $k \in \{AA, AB, BB\}$

$$p(x_i|Z_i = k, N_i) = Bin(x_i|N_i, \mu_k)$$

Note: $Z_i = k$ represents the probability of a genotype k at a locus i; Z_i is called a *latent* (*hidden*) *variable or state* corresponding to a mixture component at locus i and follows a categorical distribution with parameter π

The function takes as arguments:

- `x`, the input reference base counts $x_{1:T}$
- `N`, the input depth $N_{1:T}$
- `mu`, the binomial parameters $\mu_{1:K}$

The function returns a matrix of binomial probabilities with dimensions $T \times K$.

Use the built-in `dbinom` function for the binomial probability mass function (pmf).

```
compute.binom.lik <- function(x, N, mu){
  L <-  matrix(NA, nrow = length(x), ncol = K)
 for (i in 1:K) {
    L[,i] <- dbinom(x, size = N, prob = mu[i])
 }
  return(L)
}
```

1.1.2. Compute the binomial probabilities Use the `compute.binom.lik` function to compute the binomial probabilities for $x_{1:T}$ and $N_{1:T}$ from the input file. Use initial binomial parameters set, `mu.init`.

Save this to a variable named `obs.lik`. `obs.lik` should have dimensions $T \times K$ (i.e. 1000×3). Print the first 5 lines of `obs.lik`.

```
obs.lik <- compute.binom.lik(x, N, mu.init)
obs.lik[1:5, ]
```

```
##              [,1]         [,2]         [,3]
## [1,] 6.202536e-02 5.377387e-11 9.271746e-82
## [2,] 4.851809e-14 2.475124e-04 6.175313e-62
## [3,] 8.255780e-03 1.403009e-09 1.197446e-76
## [4,] 6.978069e-07 2.109893e-04 8.531618e-47
## [5,] 4.505405e-09 1.119110e-04 5.677071e-55
```

```
dim(obs.lik)
```

```
## [1] 1000    3
```

1.2. Compute the log likelihood function ℓ

1.2.1. Define a function, `compute.loglik`, that will compute the log likelihood function of the mixture model for the current parameter setttings

$$\ell = \sum_{i=1}^{T} \log \left(\sum_{k=1}^{K} \pi_k Bin(x_i | N_i, \mu_k) \right)$$

The function takes as arguments:

- `obs.lik`, the likelihood (binomial probabilities) computed from `compute.binom.lik`
- `pi`, the probability of the genotypes $\pi_{1:K}$

The function returns a single $\log_e$ number.

```
compute.loglik <- function(obs.lik, pi){
  wL <- 0
  l <- 0
  for (i in 1:K) {
    wL <- wL + pi[i]*obs.lik[,i]
  }
  l <- sum(log(wL))
  return(l)
}
```

1.2.2. Compute the log likelihood for the input data Use the `compute.loglik` function to compute the log likelihood for the input data. Thus, compute the log likelihood for `obs.lik` using the initial probabilities for genotypes set, `pi.init` as input values to the function.

```
loglik <- compute.loglik(obs.lik, pi.init)
loglik
```

```
## [1] -3993.978
```

```
dim(obs.lik)
```

```
## [1] 1000    3
```

2. Implement Functions for Genotype Inference and Parameter Estimation using EM

2.1 Compute the responsibilities in the E-Step (inference step)

2.1.1 Write a function to compute the responsibilities

Write a function, called `compute.responsibilities`, that computes the probability of each locus i having every possible genotype $\forall k \in \{AA, AB, BB\}$,

$$\gamma(Z_i = k) = \frac{\pi_k Bin(x_i|N_i, \mu_k)}{\sum_{j=1}^{K} \pi_j Bin(x_i|N_i, \mu_j)}$$

Note: This probability is computed for each locus i and each genotype k. In other words, what is the probability of locus i having genotype k?

This function take as arguments:

- `obs.lik`, the likelihood (binomial probabilities) computed from `compute.binom.lik`
- `pi`, the probability of the genotypes $\pi_{1:K}$

The function returns the responsibilities (expectations), the posterior distribution of latent variables, that is a matrix of binomial probabilities with dimensions $T \times K$ where the rows represent the data points (genomic loci) and the columns represent the genotypes. The denominator in the equation is the normalization constant, which leads to each row summing to 1.

```
compute.responsibilities <- function(obs.lik, pi){
  gamma <- matrix(NA, nrow = nrow(obs.lik), ncol = K)
 # print(dim(gamma))
  wL <- 0

  for (i in 1:K) {
   gamma[,i] <-  pi[i]*obs.lik[,i]    #posterior - product of prior and likelihood (Bayes' theorem)
   wL <- wL + gamma[,i]
  }

  return(gamma/wL)
}
```

2.1.2 Compute $\gamma(Z_{1:T})$ for the first EM iteration

Compute the responsibilities using the initial settings of the parameters `pi.init` and the binomial probabilities computed, `obs.lik`. This is basically the E-Step in the first iteration of EM.

Print out the log likelihood and the first 3 lines of the responsibility matrix ($\gamma(Z_{1:3})$).

```
gamma <- compute.responsibilities(obs.lik, pi.init)
gamma[1:3, ]
```

```
##              [,1]         [,2]         [,3]
## [1,] 1.000000e+00 1.625561e-10 9.342696e-82
## [2,] 1.045455e-09 1.000000e+00 8.316505e-59
## [3,] 1.000000e+00 3.186425e-08 9.065209e-76
```

```
dim(gamma)
```

```
## [1] 1000    3
```

2.2 Updating the probability of genotypes (mixture weights) in the M-Step (learning step)

2.2.1 Write a function to update the parameter $\pi_{1:K}$

Write a function, called `update.pi`, that computes the update of π_k given the responsibilities $\gamma(Z_i)$ from the E-Step and the hyperparameter δ_k

$$\hat{\pi}_k = \frac{\left(\sum_{i=1}^{T} \gamma(Z_i = k)\right) + \delta(k) - 1}{\sum_{j=1}^{K} \left\{\left(\sum_{i=1}^{T} \gamma(Z_i = j)\right) + \delta(j) - 1\right\}}$$

Note: This is the *maximum a posteriori* (MAP) estimate for the π parameter computed from the posterior distribution of latent variables (`gamma`) with the mode of a Dirichlet prior distribution parameterized by `delta`. What is the probability of genotype k?

This function takes as arguments:

- `gamma`, the responsibilities computed from `compute.responsibilities`
- `delta`, the hyperparameter for the Dirichlet prior, $\delta_{1:K}$

The function returns a vector $\hat{\pi}_{1:K}$ with K elements, one value for each genotype $k \in \{AA,\ AB,\ BB\}$.

```
update.pi <- function(gamma, delta){
  pi.hat <- rep(0, times = K)

  for (i in 1:K) {
    pi.hat[i] <- colSums(gamma, na.rm = TRUE)[i] + delta[i] - 1
  }
  pi.hat <- pi.hat/sum(pi.hat)
  return(pi.hat)
}
```

2.2.2 Compute $\pi_{1:T}$ for the first EM iteration

Use `update.pi` to compute $\hat{\pi}$ for the first iteration of EM. Use the responsibilities, `gamma` and hyperparameter for the prior of the initial state distribution, `delta.hyper`. Print out the values of $\hat{\pi}_{1:K}$.

```
pi.hat <- update.pi(gamma, delta.hyper)
pi.hat
```

```
## [1] 0.793479840 0.202007532 0.004512628
```

2.3 Updating the binomial parameter $\mu_{1:K}$ in the M-Step

2.3.1. Write a function to update the binomial parameter

Write a function, called `update.mu`, that computes the update of μ_k given the responsibilities $\gamma(Z_i)$ from the E-Step and the hyperparameters α_k and β_k

$$\hat{\mu}_k = \frac{\left(\sum_{i=1}^{T} \gamma(Z_i = k) x_i\right) + \alpha_k - 1}{\left(\sum_{i=1}^{T} \gamma(Z_i = k) N_i\right) + \alpha_k + \beta_k - 2}$$

Note: This is the *maximum a posteriori* (MAP) estimate for the μ parameter computed from the posterior distribution of latent variables (`gamma`) with the mode (maximum) of a Beta prior distribution parameterized by `alpha` and `beta`. What is the probability of observing a reference base for genotype k?

This function takes as arguments:

- the responsibilities computed from `compute.responsibilities`
- the input reference base counts $x_{1:T}$
- the input depth $N_{1:T}$
- the hyperparameters for the Beta prior, $\alpha_{1:K}$ and $\beta_{1:K}$

The function returns a vector $\hat{\mu}_{1:K}$ with K elements, one value for each genotype $k \in \{AA,\ AB,\ BB\}$.

Note: $\sum_{i=1}^{T} \gamma(Z_i = k) x_i$ can be computed using matrix multiplication. Since $\gamma(Z_{1:T})$ is a matrix with dimensions $T \times K$ and the transpose of $x_{1:T}$ (i.e. $(x_{1:T})^T$) is $1 \times T$, then $\boldsymbol{x^T} \times \boldsymbol{\gamma}$ will have dimensions $1 \times K$.

```
update.mu <- function(gamma, x, N, alpha, beta){
  mu.hat <- rep(0, times = K)
  num <- 0
  denom <- 0

  num <- t(x)%*%gamma
  denom <- t(N)%*%gamma

 for (i in 1:K) {
    num[i] <- num[i] + alpha[i] - 1
    denom[i] <- denom[i] + alpha[i] +beta[i] - 2
 }
 mu.hat <- num/denom
 return(mu.hat)
}
```

2.3.2 Compute the binomial parameter for the first EM iteration

Use `update.mu` to compute $\hat{\mu}_{1:K}$ for the first iteration of EM. Use the computed responsibilities and hyperparameters for the Beta prior, `alpha.hyper` and `beta.hyper`. Print out the values of $\hat{\mu}_{1:K}$.

```
mu.hat <- update.mu(gamma, x, N, alpha.hyper, beta.hyper)
mu.hat
```

```
##            [,1]      [,2]      [,3]
## [1,] 0.9653329 0.6410865 0.1194937
```

2.4. Compute the log posterior

2.4.1. Define a function, `compute.log.posterior`, that will compute the log posterior distribution

$$\log \mathbb{P} = \ell + \log Dirichlet(\hat{\boldsymbol{\pi}}|\boldsymbol{\delta}) + \sum_{k=1}^{K} \log Beta(\hat{\mu}_k|\alpha_k, \beta_k)$$

Log posterior is the summation of log and log priors

The function takes as argmuents:

- `loglik`, the log likelihood ℓ computed from `compute.loglik`
- `mu`, the binomial parameters $\mu_{1:K}$
- `pi`, the probability of the genotypes $\pi_{1:K}$
- `delta`, the hyperparameter for the Dirichlet prior, $\delta_{1:K}$,
- `beta`, the hyperparameters for the Beta prior, $\alpha_{1:K}$ and $\beta_{1:K}$

```r
compute.log.posterior <- function(loglik, mu, pi, alpha, beta, delta){
  return(sum(loglik, dirichletpdflog(pi, delta), sum(betapdflog(mu,alpha, beta)) ))
}
```

Note: can use the R function `dbeta` for the Beta distribution or `betapdflog` below. For the Dirichlet distribution, use the function below. `dirichletpdflog` accepts input vectors `pi` and `delta`, both of length `K`. The prior probabilities in the log posterior should be in log space.

```r
dirichletpdflog <- function(pi, delta) {
    c <- lgamma(sum(delta, na.rm = TRUE)) - sum(lgamma(delta), na.rm = TRUE)  #normalizing constant
    l <- sum((delta - 1) * log(pi), na.rm = TRUE)  #likelihood
    return(c + l)
}

betapdflog <- function(x, alpha, beta){
  c <- -lbeta(alpha, beta)
  l <- (alpha - 1) * log(x) + (beta - 1) * log(1 - x)
  return(c + l)
}
```

2.4.2. Compute the log posterior for the input data Use the `compute.log.posterior` function to compute the log posterior for the input data. Use the updated parameters, `pi.hat` and `mu.hat`. Use the hyperparameters, `delta.hyper`, `alpha.hyper`, and `beta.hyper` for the priors.

First recompute the `obs.lik` using `compute.binom.lik` and then the `loglik` using `compute.loglik` for the updated parameters, `mu.hat` and `pi.ha`.

```r
obs.lik <- compute.binom.lik(x, N, mu.hat)
loglik <- compute.loglik(obs.lik, pi.hat)
logP <- compute.log.posterior(loglik, mu.hat, pi.hat, alpha.hyper, beta.hyper, delta.hyper)
logP
```

```
## [1] -2725.843
```

The function `compute.log.posterior` returns a single value that will be used to monitor the EM algorithm at each iteration for convergence.

3. Run the Full EM algorithm

3.1 Implement the full EM algorithm

Implement the full EM algorithm for inferring the responsibilities and learning the binomial mixture model parameters in a Bayesian framework.

Run the EM algorithm until convergence, compute.log.posterior. Convergence is achieved when the log posterior changes by less than a user-defined valued, $\epsilon = 10^{-2}$.

i. Print out the converged parameters:

   - the converged parameters for the probability of the genotypes $\hat{\pi}_k$ for each genotype $k \in \{AA, AB, BB\}$.
   - the converged parameters for the probability of observing a reference base $\hat{\mu}_k$ for each genotype $k \in \{AA, AB, BB\}$.

ii. Save the the responsibilities $\gamma(Z_i)$ for the final EM iteration. Print out the responsibilities of the first 5 loci for the final EM iteration

iii. Store the log posterior for each iteration of EM. Print out the log posterior for all iterations.

```
converged = FALSE
epsilon = 10^-2

#Initialize
pi <- pi.init
mu <- mu.init
logP <-  -Inf
curr.iter <- 1
prev.iter <- 0

#Compute the observed likelihood using initial parameters
obs.lik <- compute.binom.lik(x, N, mu)

while (converged == FALSE) {

  curr.iter <-  curr.iter + 1

  #E-step: Compute responsibilities
  gamma <- compute.responsibilities(obs.lik, pi)

  #M-step: Update parameters
  pi.hat <- update.pi(gamma, delta.hyper)
  mu.hat <- update.mu(gamma, x, N, alpha.hyper, beta.hyper)

  #M-step: Assign updated parameters
  pi <- pi.hat
  mu <- mu.hat

  #M-step: Recompute the observed likelihood using update parameters
  obs.lik <- compute.binom.lik(x, N, mu)

  #M-step:Compute the log likelihood
  loglik <- compute.loglik(obs.lik, pi)
  #M-step:Compute log Posterior
  logP[curr.iter] <- compute.log.posterior(loglik, mu, pi, alpha.hyper, beta.hyper, delta.hyper)
  #print(logP[curr.iter])

  prev.iter <- curr.iter - 1

  if ( (logP[curr.iter] - logP[prev.iter])  < epsilon ) {
    converged = TRUE
  }
  logP[prev.iter] <-  logP[curr.iter]
}

print(pi)
```

```
## [1] 0.75145895 0.19502661 0.05351445
```

```
print(mu)
```

```
##           [,1]      [,2]      [,3]
## [1,] 0.9705156 0.7817451 0.2640394
```

```
print(gamma[1:5,])
```

```
##              [,1]        [,2]         [,3]
## [1,] 9.984052e-01 0.001594753 7.977912e-23
## [2,] 3.464552e-07 0.999999654 6.350446e-12
## [3,] 9.820998e-01 0.017900187 2.643911e-20
## [4,] 6.492569e-03 0.993507431 2.226454e-10
## [5,] 3.520319e-04 0.999647968 1.119168e-11
```

```
print(logP[1:prev.iter])
```

```
## [1] -2725.843 -2361.785 -2299.422 -2293.413 -2292.666 -2292.570 -2292.557
## [8] -2292.555
```

3.2 Determine the mutation status for the input data

Using the responsibilities `gamma` from the final EM iteration, determine the mutation status (`NotSNV` or `SNV`) for each locus.

- Loci i is assigned `SNV` when $\gamma(Z_i = AB) + \gamma(Z_i = BB) >= 0.9$
- Loci i is assigned `NotSNV` when $\gamma(Z_i = AB) + \gamma(Z_i = BB) < 0.9$

Note: To call a variant for each locus i, a threshold is applied on the responsibilities $\gamma(Z_i)$. Summing $\gamma(Z_i = AB)$ and $\gamma(Z_i = BB)$ gives the overall probability (either genotype AB or BB) that locus i is a variant containing the non-reference allele (B)

Append this to the original table. Save the results to a file, `mutationCalls_LAC.txt`, with the same columns as the input file (`counts`) and then appending an additional column with the mutation status (`NotSNV` or `SNV`)

```
* `chr`, `position`, `refBase`, `refCount`, `NRefBase`, `NrefCount`, `depth`
* `mutationStatus`
```

```r
library(tidyverse)

gamma.EM.converged <- data.frame(gamma)
colnames(gamma.EM.converged) <-  c("AA","AB", "BB")
gamma.EM.converged <- tibble(gamma.EM.converged)

gamma.mut <- gamma.EM.converged
gamma.mut <- gamma.EM.converged %>%
  mutate(mutationStatus = case_when((AB + BB >= 0.9) ~ "SNV", (AB + BB < 0.9) ~ "NotSNV")) %>%
  select(-AA, -AB, -BB) %>%
  mutate(id = seq.int(1, nrow(gamma.mut)))

counts.tibble <- as.tibble(counts)
counts.tibble <- as.tibble(counts) %>%
  mutate(id = seq.int(1, nrow(counts.tibble)))

mutationCalls <- counts.tibble %>%
  left_join(gamma.mut, by = "id") %>%
  select(-id)

mutationCalls[1:5,]
```

```
## # A tibble: 5 x 8
##     chr position refBase refCount NRefBase NrefCount depth mutationStatus
##   <int>    <int> <chr>      <int> <chr>        <int> <int> <chr>
## 1     2    91822 C             42 N                2    44 NotSNV
## 2     2   202466 C             36 N               12    48 SNV
## 3     2   274584 C             40 N                3    43 NotSNV
## 4     2   756080 T             26 N                6    32 SNV
## 5     2  1052968 C             31 N                8    39 SNV
```

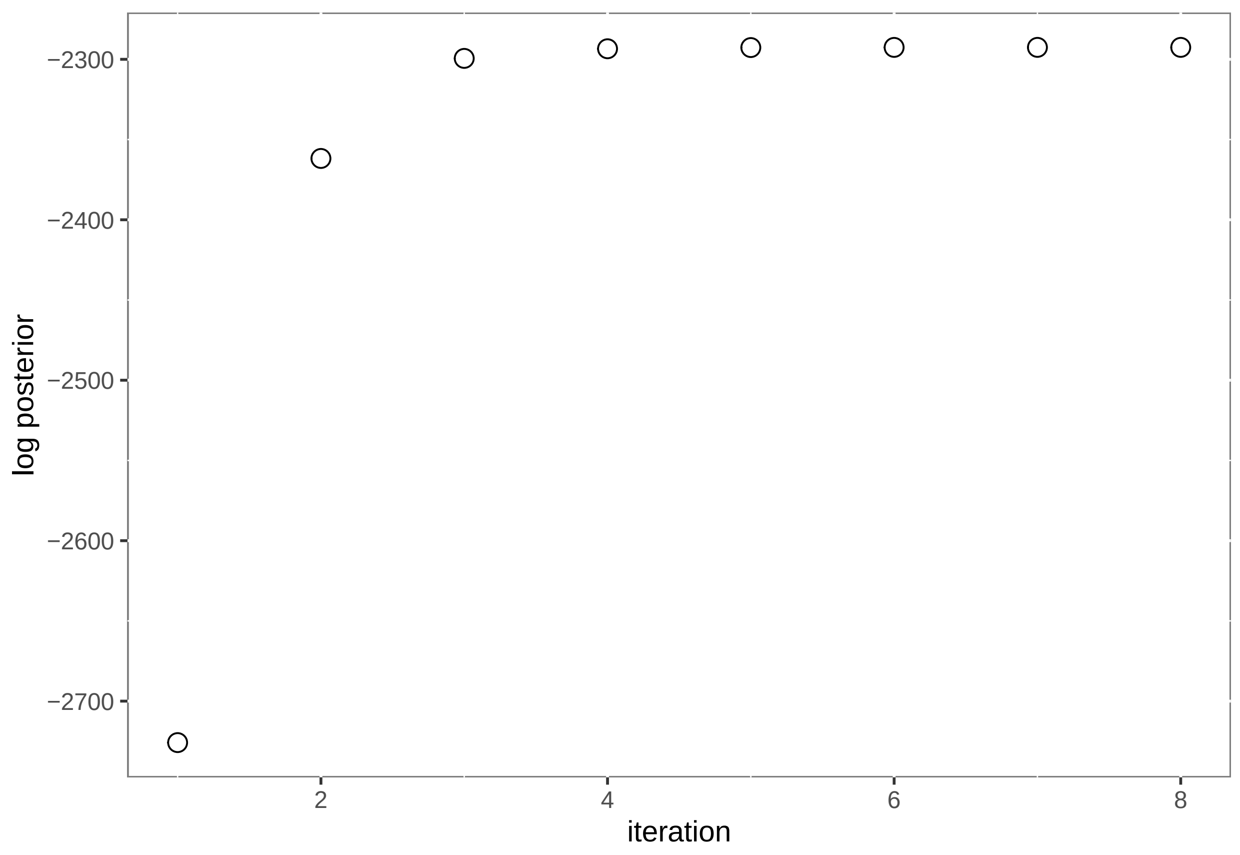
```r
write_delim(mutationCalls,"mutationsCalls_LAC.txt")
```

3.3. Plot the log posterior

Plot the log posterior as a function of the EM iterations. The plot should show a monotonically increasing log posterior curve, which is a property of the EM algorithm. The x-axis should be the EM iteration number and y-axis is the log posterior.

```r
#plot(logP[1:prev.iter])
df <- data.frame(cbind(seq.int(1,prev.iter), logP[1:prev.iter]))
colnames(df) <- c("iteration", "logP")

ggplot(df) +
  geom_point(mapping = aes(x=iteration,y=logP), size = 3, shape = 1, color = "black") +
  labs(x= "iteration", y="log posterior") +
  theme(panel.background = element_rect(fill = "white", colour = "grey50"))
```

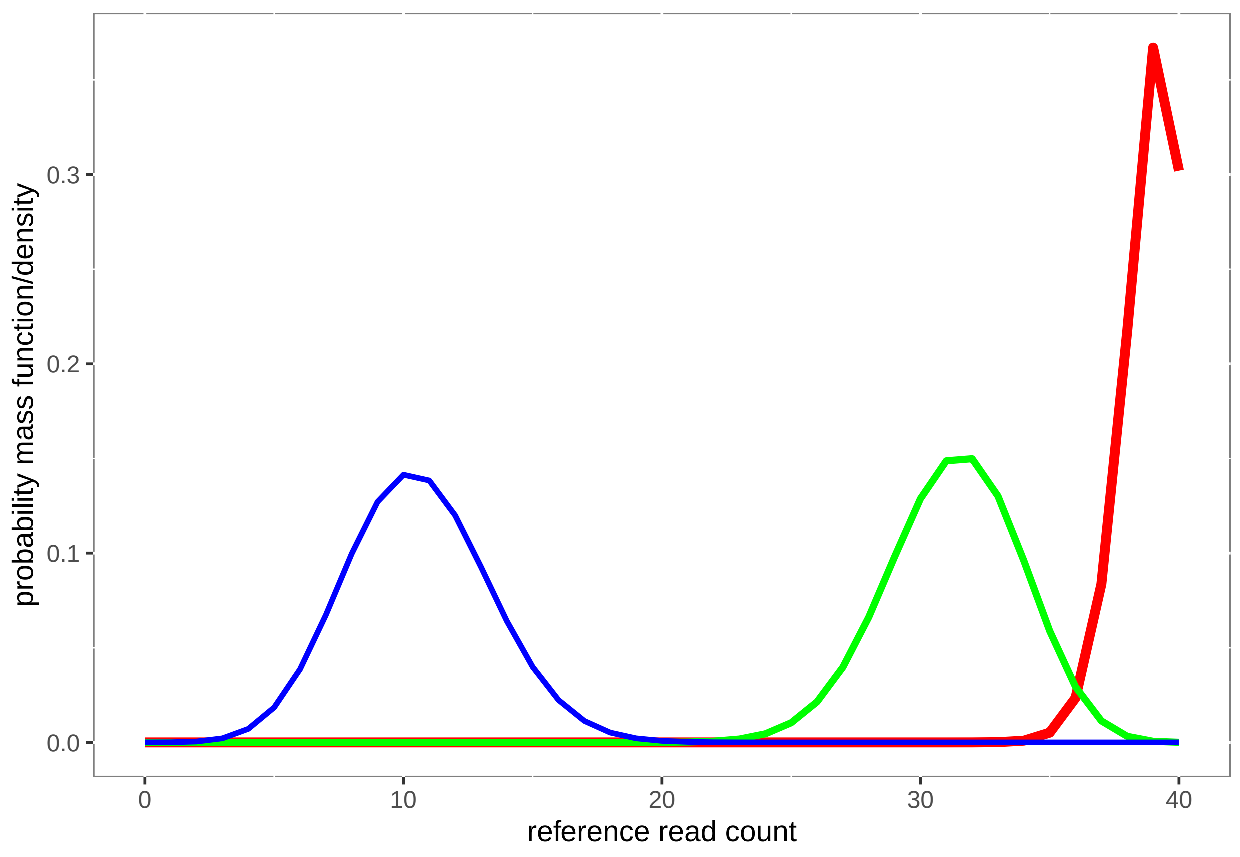


3.4. Plot the binomial pmf for the converged parameters

Plot the probability mass function (pmf) for each binomial distribution in the 3-component mixture using the converged parameters $\hat{\mu}_{1:K}$. Use $N = 40$ depth and a range of values $x = \{0, \ldots, 40\}$ reference read counts.

```
ref.read.count <- seq.int(0,40)
depth <- 40
mu.em.converged <- mu
pmf <- data.frame(compute.binom.lik(ref.read.count, depth, mu.em.converged))
colnames(pmf) <- c("AA","AB", "BB")

ggplot(pmf) +
  geom_line(mapping = aes(x=ref.read.count,y=AA), colour ="red", size =1.75) +
  geom_line(mapping = aes(x=ref.read.count,y=AB), colour ="green", size =1.25) +
  geom_line(mapping = aes(x=ref.read.count,y=BB), colour ="blue",size =1) +
  labs(x= "reference read count", y="probability mass function/density") +
  theme(panel.background = element_rect(fill = "white", colour = "grey50"))
```



probability mass function/density
0.3
0.2
0.1
0.0
0
10
20
30
40
reference read count