The EM algorithm

Computational Statistics

Adam Gorm Hoffmann

Setup where the EM algorithm is useful

- We observe incomplete data $X = M(Y)$.
- Log-likelihood $\log g(x \mid \theta)$ is impossible to find analytically, and expensive to compute numerically.
- Complete data log-likelihood $\log f(y \mid \theta)$ may be cheap to compute, but we don't observe y, only the incomplete data $x = M(y)$.
- It is cheap to compute $Q(\theta \mid \theta') = E_{\theta'}(\log f(Y \mid \theta) \mid X = x)$, the expected complete data log-likelihood in θ conditional on our incomplete data computed under the distribution $f_{\theta'}$.

The EM algorithm

Idea: Start with initial guess θ_0 and iteratively compute

$$\theta_{n+1} = \arg\max_{\theta} Q(\theta \mid \theta_n).$$

Stop when some convergence criterion is reached (*e.g.*, after a specific number of iterations, or when $\|\theta_{n+1} - \theta_n\|$ is sufficiently small).

Why "EM"? For a given θ_n:

- **E**xpectation step: Compute the function $\theta \mapsto Q(\theta \mid \theta_n)$.
- **M**aximization step: Maximize $\theta \mapsto Q(\theta \mid \theta_n)$.

Each EM iteration increases the log-likelihood

Denote the cross-entropy $H(\theta \mid \theta') = E_{\theta'}(-\log h(Y \mid x, \theta) \mid X = x)$.

If $Q(\theta \mid \theta') > Q(\theta' \mid \theta')$, then

$$\begin{aligned}
\ell(\theta) &= Q(\theta \mid \theta') + H(\theta \mid \theta') \\
&\geq Q(\theta \mid \theta') + H(\theta' \mid \theta') && \text{Gibbs' inequality} \\
&> Q(\theta' \mid \theta') + H(\theta' \mid \theta') && \text{Assumption} \\
&= \ell(\theta').
\end{aligned}$$

General R implementation

```r
## Either Q or update_estimates must be supplied. If Q is supplied,
## numerical optimization is used to update the estimates.
EM <- function(epsilon, par_init, update_estimates = NULL, Q = NULL,
               callback = NULL)
{
    if (is.null(update_estimates))
        update_estimates <- get_update_estimates_general(Q)
    epsilon_squared <- epsilon^2
    new_est <- par_init
    not_converged <- TRUE
    while (not_converged) {
        old_est <- new_est
        new_est <- update_estimates(old_est)
        not_converged <- crossprod(new_est - old_est) >
            epsilon_squared * (crossprod(old_est) + epsilon)^2
        if (!is.null(callback))
            callback()
    }
    new_est
}
```

M step for general Q with `optim`

```r
get_update_estimates_general <- function(Q)
{
    force_all_args()
    function(old_est)
        optim(old_est, function(par) -Q(par, old_est),
              control = list(reltol = 1e-16))$par
}

## Forces evaluation of all arguments
force_all_args <- function()
    as.list(parent.frame())
```

Specific problem setup

Let X follow the generalized t-distribution

$$f(x) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi\sigma^2}\Gamma\left(\frac{\nu}{2}\right)}\left(1 + \frac{(x-\mu)^2}{\nu\sigma^2}\right)^{-\frac{\nu+1}{2}}.$$

There is no analytical solution for the MLE $(\hat{\mu}, \hat{\sigma}^2)$.

If $W \sim \chi^2_\nu$ and $X \mid W = w \sim N(\mu, \nu\sigma^2/w)$ then X follows the desired generalized t-distribution.

We can use the EM algorithm in this setup!

Specific problem setup

$Y = (X, W) \in \mathbb{R} \times (0, \infty)$ has joint density

$$f(x, w) = f(x \mid w) f(w) = \frac{1}{\sqrt{\pi \nu \sigma^2} 2^{(\nu+1)/2} \Gamma(\nu/2)} w^{\frac{\nu-1}{2}} e^{-\frac{w}{2}\left(1 + \frac{(x-\mu)^2}{\nu \sigma^2}\right)}$$

with fixed $\nu \in (0, \infty)$ and unknown parameters $\mu \in \mathbb{R}, \sigma^2 \in (0, \infty)$.

Goal: use the EM algorithm to estimate (μ, σ^2) when we only observe iid. $X_1, \ldots, X_n$.

Simulating from the model

```r
sim_y <- function(n, mu, sigma2, nu)
{
    w <- rchisq(n, df = nu)
    x <- rnorm(n, mean = mu, sd = sqrt(nu * sigma2 / w))
    list(
        x = x,
        w = w
    )
}
```

Numerical optimization in Stan

- Stan has a fast optimizer using LBFGS, BFGS or Newton.
- The marginal distribution as a Stan model:

```
data {
  int<lower = 1> N;
  real x[N];
  int<lower = 1> nu;
}
parameters {
  real mu;
  real<lower = 0> sigma;
}
model {
  x ~ student_t(nu, mu, sigma);
}
```

Running the numerical optimization

```r
stan_marginal <- stan_model(file = "marginal.stan")
stan_opt <- optimizing(stan_marginal,
                       data = list(N = n,
                                   x = y$x,
                                   nu = nu),
                       init = list(mu = 0, sigma = 1))
```

- Done instantly (3 ms when $n = 1000$, 900 ms when $n = 10^6$).
- We can use this to check that our implementation works.

The E step (deriving Q)

Up to an additive constant

$$Q(\mu, \sigma^2 \mid \mu_m, \sigma_m^2) = -\frac{n}{2} \log \sigma^2 - \frac{1}{2\nu\sigma^2} \sum_{i=1}^{n} (x_i - \mu)^2 \frac{1+\nu}{1 + \frac{(x_i - \mu_m)^2}{\nu \sigma_m^2}}.$$

The M Step (maximizing Q)

Solving $\frac{d}{d\mu} Q(\mu, \sigma^2 \mid \mu_m, \sigma_m^2) = 0$ and $\frac{d}{d\sigma^2} Q(\mu, \sigma^2 \mid \mu_m, \sigma_m^2) = 0$ yields

$$\mu = \frac{\sum_{i=1}^{n} \frac{x_i}{1+\frac{(x_i-\mu_m)^2}{\nu\sigma_m^2}}}{\sum_{i=1}^{n} \frac{1}{1+\frac{(x_i-\mu_m)^2}{\nu\sigma_m^2}}}$$

$$\sigma^2 = \frac{1}{n\nu} \sum_{i=1}^{n} (x_i - \mu)^2 \frac{1 + nu}{1 + \frac{(x_i-\mu_m)^2}{\nu\sigma_m^2}}$$

Since Q is concave, this is a global maximum.

R implementation of Q

```r
## Only implemented up to additive constant,
## since we only need the derivatives.
getQ <- function(x, nu)
{
    force_all_args()
    function(par, par_prime)
        - length(x) * log(par[2]) / 2 -
            sum((x - par[1])^2 /
                (1 + (x - par_prime[1])^2 / (nu * par_prime[2]))) *
            (1 + nu) / (2 * nu * par[2])
}
```

Profiling: General implementation

- Evaluating Q in `optim` is the only part that takes significant time.
- **Idea**: do specific implementation of M step.

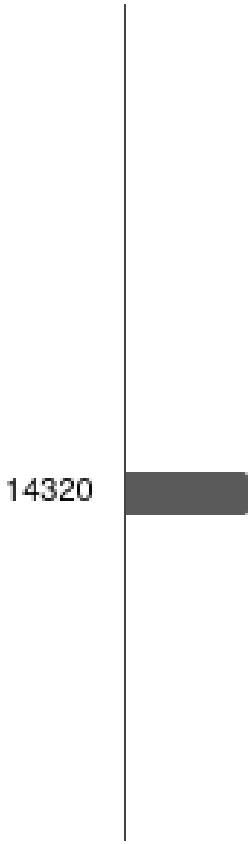
R implementation of EM step

```r
get_update_estimates_specific <- function(x, nu)
{
    force_all_args()
    n <- length(x)
    function(old_est)
    {
        beta_inverse <- (1 + (x - old_est[1])^2 / (nu * old_est[2]))^(-1)
        mu_new <- sum(x * beta_inverse) / sum(beta_inverse)
        sigma2_new <- sum((x - mu_new)^2 *
                          (1 + nu) * beta_inverse) / (n * nu)
        c(mu = mu_new, sigma2 = sigma2_new)
    }
}
```

Profiling: Specific implementation

- Updating the estimates is the only part that takes significant time.

```r
EM <- function(epsilon, par_init, update_estimates = NULL, Q = NULL,
               callback = NULL)
{
    if (is.null(update_estimates))
        update_estimates <- get_update_estimates_general(Q)
    epsilon_squared <- epsilon^2
    new_est <- par_init
    not_converged <- TRUE
    while (not_converged) {
        old_est <- new_est
        new_est <- update_estimates(old_est)
        not_converged <- crossprod(new_est - old_est) >
            epsilon_squared * (crossprod(old_est) + epsilon)^2
        if (!is.null(callback))
            callback()
    }
    new_est
}
```



Profiling: Specific implementation

```r
get_update_estimates_specific <- function(x, nu)
{
    force(x)
    force(nu)
    function(old_est)
    {
        beta_inverse <- (1 + (x - old_est[1])^2 / (nu * old_est[2]))^(-1)        11540
        mu_new <- sum(x * beta_inverse) / sum(beta_inverse)                       2400
        sigma2_new <- sum((x - mu_new)^2 * (1 + nu) * beta_inverse) / (n * nu)    2660
        c(mu = mu_new, sigma2 = sigma2_new)
    }
}
```

- Everything is already vectorized.
- **Idea**: implement in Rcpp to avoid wasting time jumping between R and low-level libraries.

Rcpp implementation: `estimates` struct

```
struct estimates {
  double mu;
  double sigma2;
};

estimates
make_estimates(double mu, double sigma2)
{
  estimates est;
  est.mu = mu;
  est.sigma2 = sigma2;
  return est;
}
```

Rcpp implementation: highest level

```
estimates
EM(double epsilon, const NumericVector& x, double nu,
   double mu_guess, double sigma2_guess)
{
  double epsilon_squared = square(epsilon);
  estimates old_est;
  estimates new_est = make_estimates(mu_guess, sigma2_guess);
  do {
    old_est = new_est;
    new_est = update_estimates(old_est, x, nu);
  } while (diff_two_norm(new_est, old_est)
           > epsilon_squared * square(two_norm(old_est) + epsilon));
  return new_est;
}

// [[Rcpp::export]]
NumericVector
EM_cpp(double epsilon, const NumericVector& x, double nu,
       double mu_guess, double sigma2_guess)
{
  estimates est = EM(epsilon, x, nu, mu_guess, sigma2_guess);
  NumericVector est_vec = {est.mu, est.sigma2};
  return est_vec;
}
```

Rcpp implementation: EM step

```
estimates
update_estimates(estimates old_estimates, const NumericVector& x, double nu)
{
  estimates new_estimates = make_estimates(0, 0);
  double mu_divide_by = 0;
  unsigned long long n_x = x.length();
  double beta_inverse[n_x];

  for (unsigned long long i = 0; i < n_x; i++) {
    beta_inverse[i] = 1 / (1 + square((x[i] - old_estimates.mu))
                        / (nu * old_estimates.sigma2));
    new_estimates.mu += x[i] * beta_inverse[i];
    mu_divide_by += beta_inverse[i];
  }
  new_estimates.mu /= mu_divide_by;
  for (unsigned long long i = 0; i < n_x; i++)
    new_estimates.sigma2 += square((x[i] - new_estimates.mu))
      * (1 + nu) * beta_inverse[i];
  new_estimates.sigma2 /= (n_x * nu);
  return new_estimates;
}
```

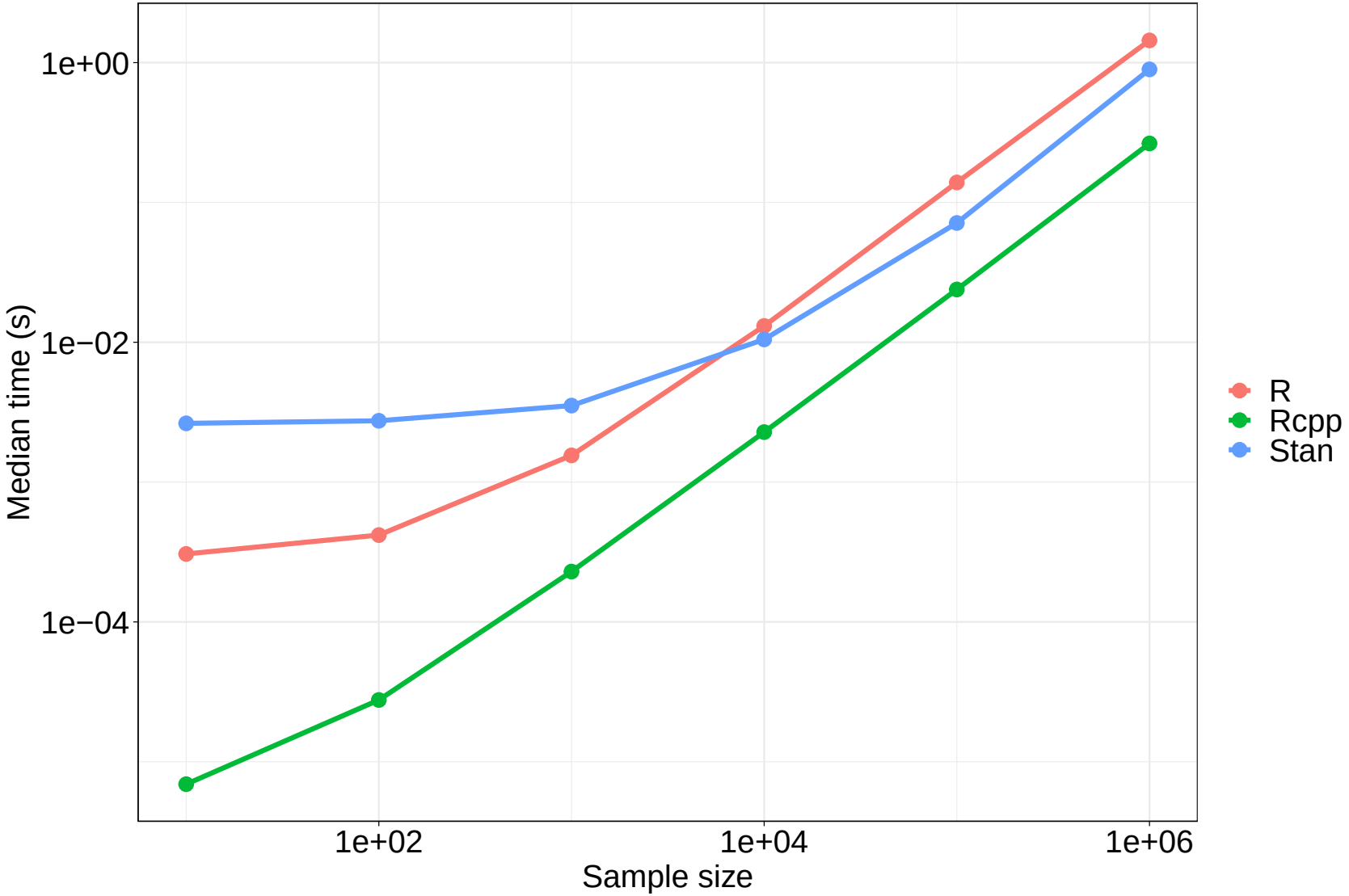
Rcpp implementation: small functions

```
double
square(double x)
{
  return x * x;
}

double
two_norm(estimates est)
{
  return square(est.mu) + square(est.sigma2);
}

double
diff_two_norm(estimates new_est, estimates old_est)
{
  return square(new_est.mu - old_est.mu)
    + square(new_est.sigma2 - old_est.sigma2);
}
```

Benchmarking EM implementations and Stan



- The Rcpp implementation is roughly 10 times faster than R, and also beats Stan's numerical optimization.

Gradient ascent: Marginal log-likelihood and gradient

```r
get_marginal_loglik <- function(x, nu)
{
    force_all_args()
    function(par)
        -n/2 * log(par[2]) -
            (nu + 1)/2 * sum(log(1 + (x - par[1])^2 / (nu * par[2])))
}

get_marginal_grad <- function(x, nu)
{
    force_all_args()
    function(par)
        c(
            (nu + 1) * sum((x - par[1]) / ((x - par[1])^2 + nu * par[2])),
            nu/2 * sum( ((x - par[1])^2 - par[2]) /
                        (par[2] * (nu * par[2] + (x - par[1])^2)) )
        )
}
```

Gradient ascent

```r
grad_ascent <- function(init_guess, objective, grad,
                        gamma = 0.01, epsilon = 1e-16,
                        callback = NULL)
{
    small_relative_ascent <- function(new_est, old_est)
    {
        objective_diff <- objective(new_est) - objective(old_est)
        objective_diff >= 0 &&
            objective_diff <= epsilon * (abs(objective(new_est)) + epsilon))
    }

    converged <- FALSE
    new_est <- init_guess
    while (!converged) {
        old_est <- new_est
        gr <- grad(old_est)
        new_est <- old_est + gamma * gr
        converged <- small_relative_ascent(new_est, old_est)
        if (!is.null(callback))
            callback()
    }

    new_est
}
```

MLE from complete data log-likelihood

If, in addition to $X_1, \ldots, X_n$, we also observe $W_1, \ldots, W_n$, then

$$\hat{\mu} = \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i},$$
$$\hat{\sigma}^2 = \frac{1}{n\nu} \sum_{i=1}^n w_i \left(x_i - \hat{\mu}\right)^2.$$

```r
mle <- function(x, w, nu)
{
    mu_hat <- sum(w * x) / sum(w)
    c(mu_hat = mu_hat,
      sigma2_hat = mean(w * (x - mu_hat)^2) / nu)
}
```

Testing that the implementations work

```r
## True parameters
mu <- 27
sigma2 <- 13
nu <- 5

## Simulating data
n <- 1000
y <- sim_y(n, mu, sigma2, nu)

## Running EM algorithm
update_estimates_specific <- get_update_estimates_specific(y$x, nu)
Q <- getQ(y$x, nu)
est_specific <- EM(1e-16, c(1,1),
                   update_estimates = update_estimates_specific)
est_general <- EM(1e-16, c(1,1), Q = Q)
est_Rcpp <- EM_cpp(1e-16, y$x, nu, 1, 1)

## Running gradient ascent
marginal_loglik <- get_marginal_loglik(y$x, nu)
marginal_grad <- get_marginal_grad(y$x, nu)
grad_ascent(c(1, 1), marginal_loglik, marginal_grad)

## Calculate complete data MLE
est_mle <- mle(y$x, y$w, nu)
```

Results of test

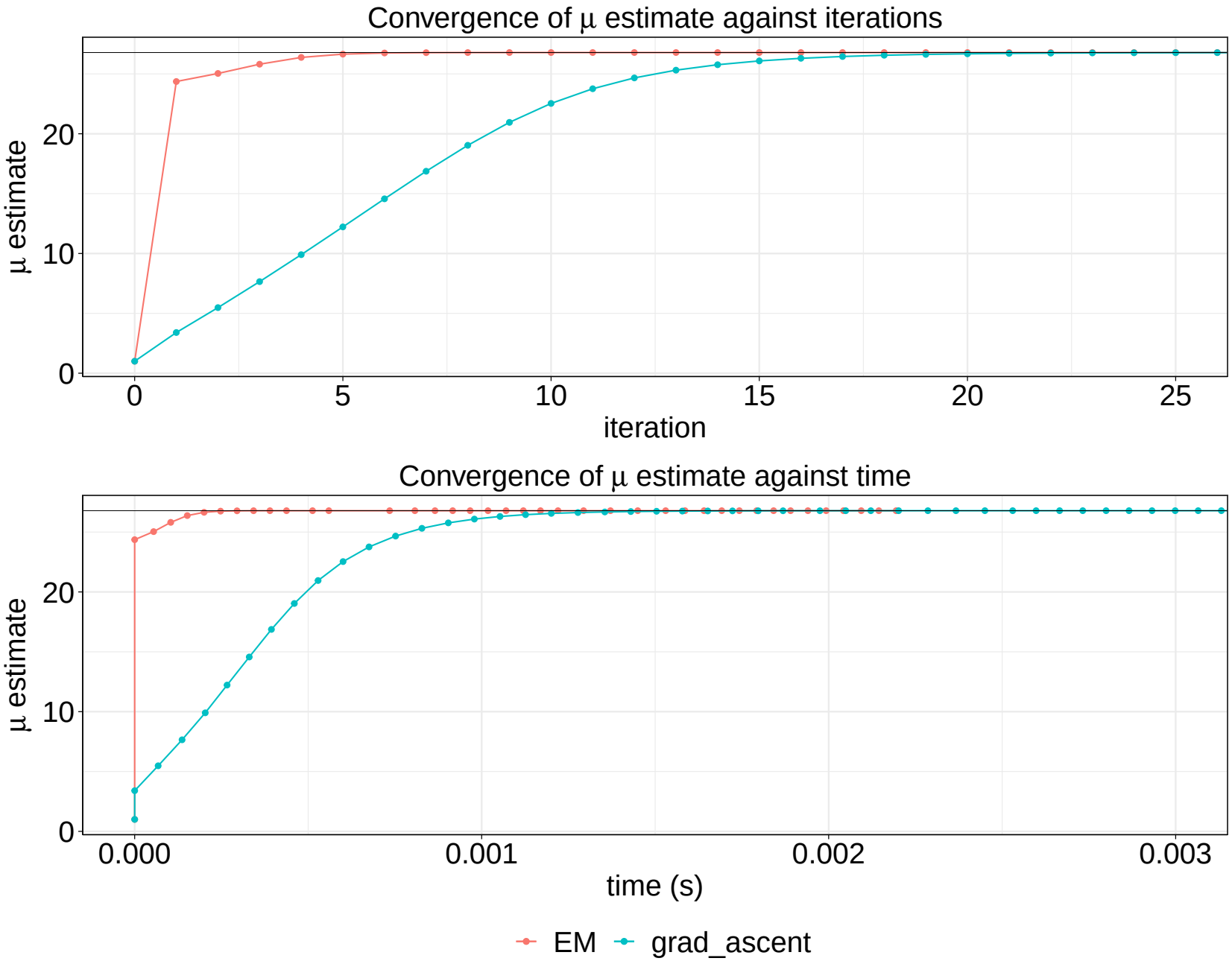
- All EM implementations, the gradient ascent implementation, and Stan give the estimate $\tilde{\mu} = 26.78594$, $\tilde{\sigma}^2 = 13.10174$ up to 7 significant digits.
- The differences between the specific R and specific Rcpp estimates are of order 10^{-15} and the differences to the more general estimate and the gradient ascent estimate are of order 10^{-10} to 10^{-6}.
- The complete data MLE is $\hat{\mu} = 26.83363$ and $\hat{\sigma}^2 = 13.21392$, so the marginal data MLE (from the EM algorithm and gradient ascent) seems reasonable.
- Since Stan agrees with my EM implementations and gradient ascent implementation, they must be correct.

Comparing convergence of EM and gradient ascent

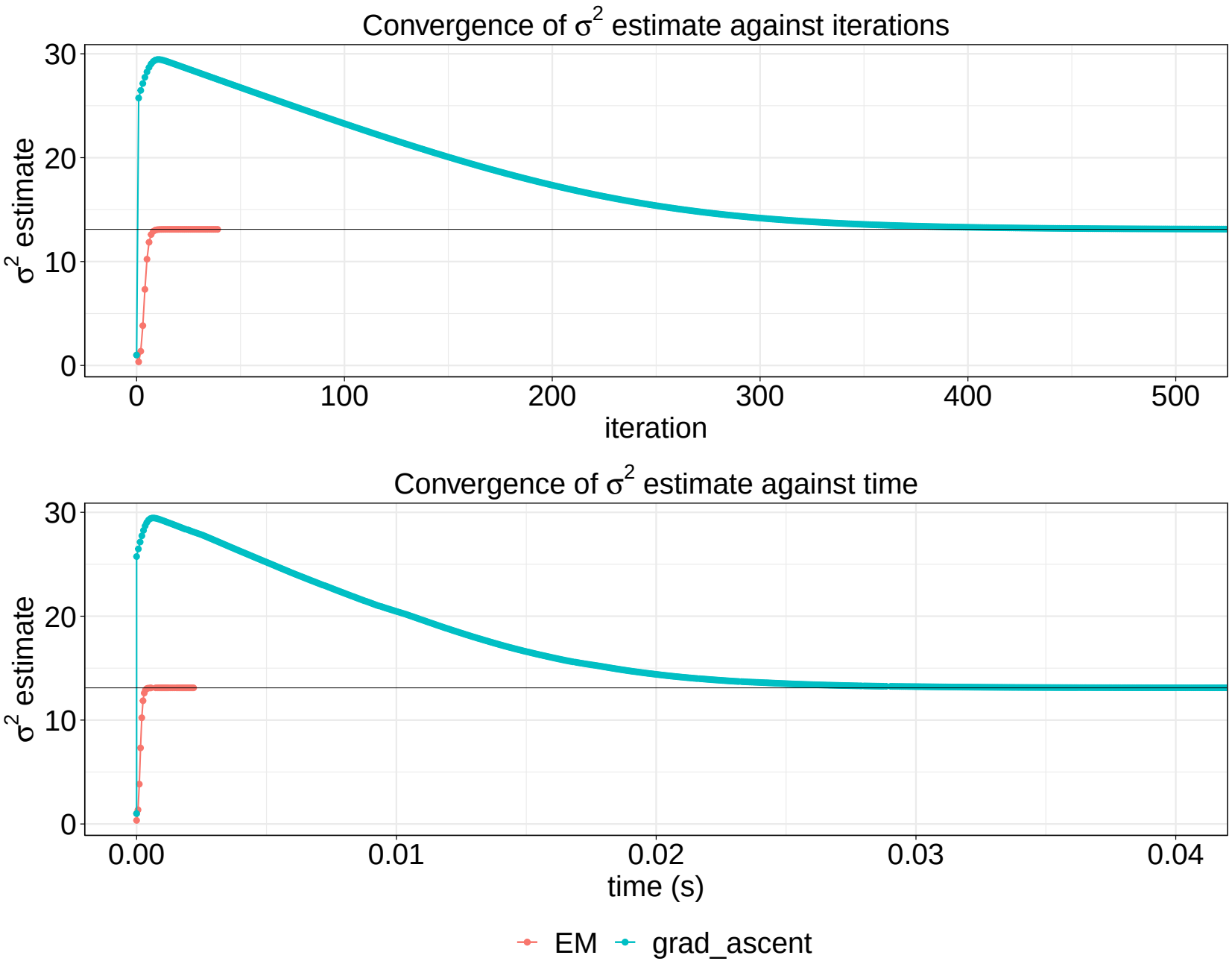
```r
em_tracer <- tracer(c("old_est", "new_est"))
EM(1e-16, c(1,1), update_estimates = update_estimates_specific,
   callback = em_tracer$tracer)

grad_tracer <- tracer(c("old_est", "new_est"))
grad_ascent(c(1, 1), marginal_loglik, marginal_grad,
            callback = grad_tracer$tracer)
```

Convergence of μ estimate



Convergence of σ^2 estimate



Linear convergence

If

$$\limsup_{n\to\infty} \frac{\|\theta_n - \theta_\infty\|}{\|\theta_{n-1} - \theta_\infty\|} = r$$

for some $r \in (0, 1)$, then we say that the convergence is linear with asymptotic rate r.

r close to 0 means large improvements in each step, and r close to 1 means small improvements in each step.

Estimating the convergence rate

If the linear convergence is reached at n_0, then (simplified)

$$\|\theta_n - \theta_\infty\| \approx r\|\theta_{n-1} - \theta_\infty\| \approx \cdots \approx r^{n-n_0}\|\theta_{n-n_0} - \theta_\infty\|,$$

so

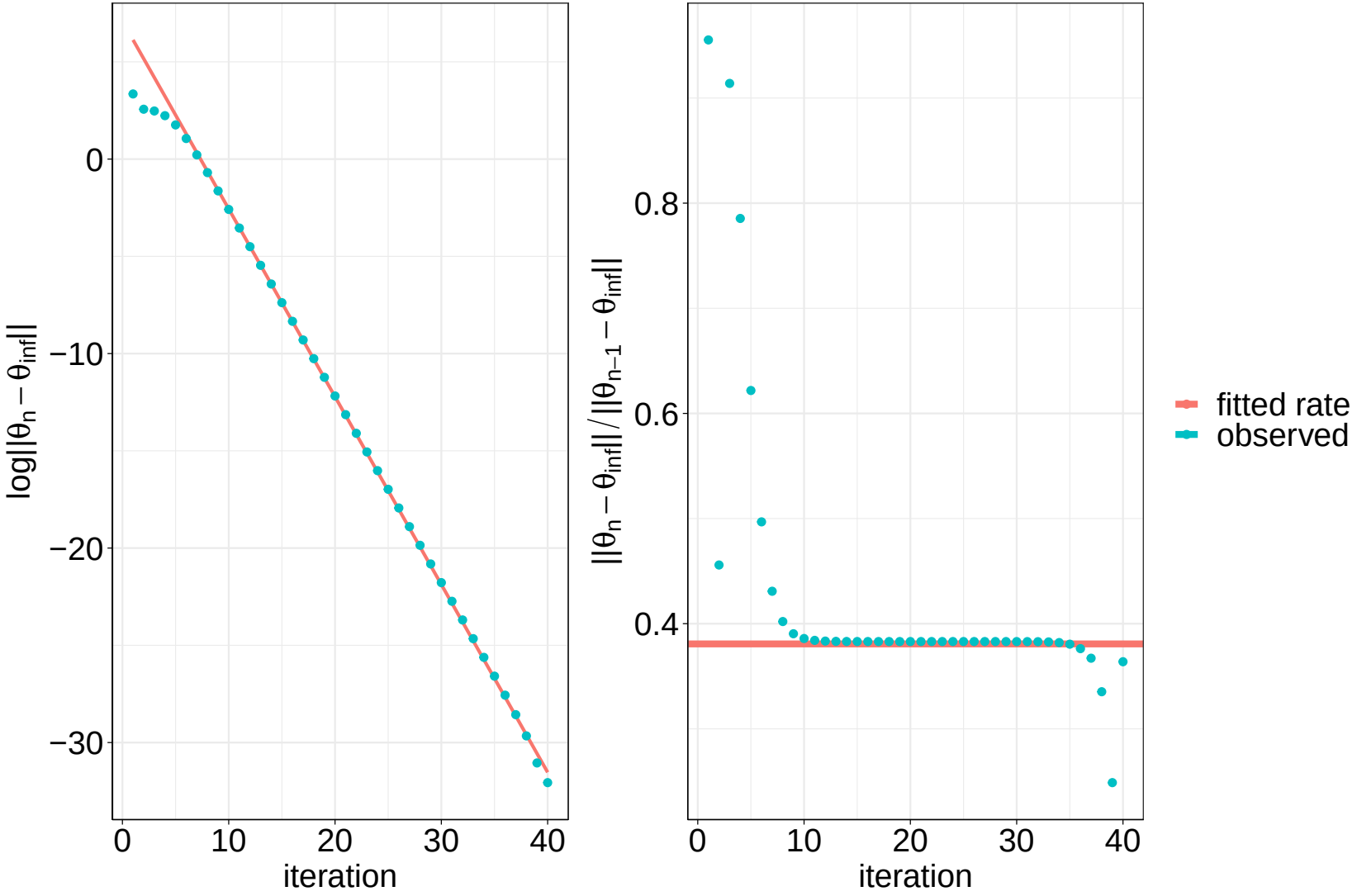
$$\log \|\theta_n - \theta_\infty\| \approx (n - n_0)\log(r) + \|\theta_{n-n_0} - \theta_\infty\|,$$

hence we can estimate $\log(r)$ as β when fitting the line

$$\log \|\theta_n - \theta_\infty\| = n\beta + \alpha.$$

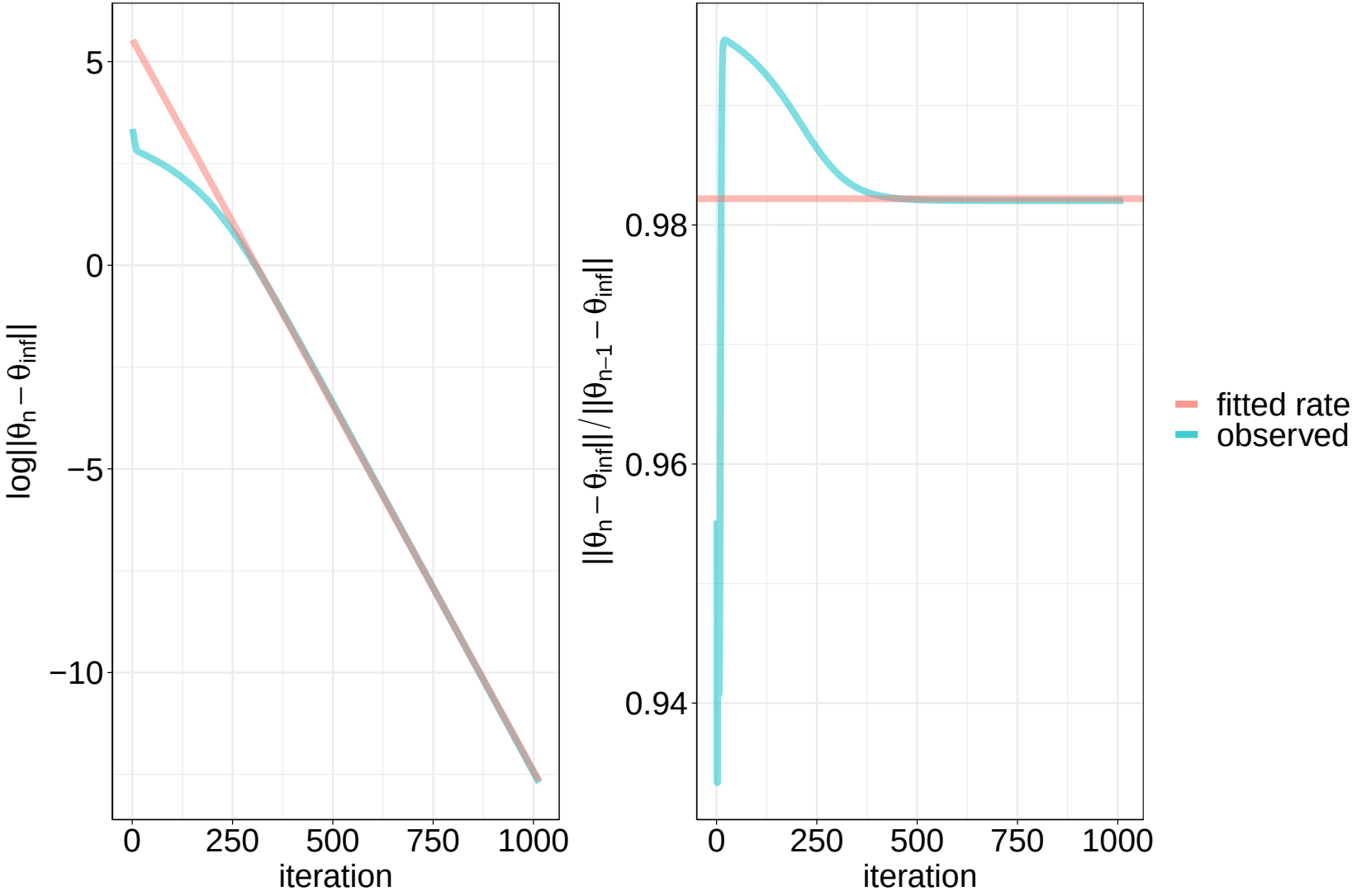
Convergence rate of EM

Linear convergence with $\hat{r} = \exp(\mathrm{slope}) \approx 0.38$.



Convergence rate of gradient ascent

Linear convergence with $\hat{r} = \exp(\mathrm{slope}) \approx 0.98$.



Conclusion from convergence investigation

- The EM algorithm converges in fewer steps and has better convergence rate than gradient ascent in our setup.
  - However, I have not spent much time tuning the parameters for gradient ascent, so it might be possible to do much better.
- My EM algorithm R implementation also converges in much less time than my gradient ascent R implementation.

Fisher information formulas

Let $\hat{i}_X = -D_\theta^2 \ell(\hat{\theta})$ denote the observed Fisher information.

Then

$$\begin{aligned}
\hat{i}_X &= -\, D_{\bar{\theta}} \left(\nabla_\theta Q(\bar{\theta} \mid \bar{\theta}) \right) \Big|_{\bar{\theta}=\hat{\theta}} \\
\hat{i}_X &= -D_\theta^2 Q(\hat{\theta} \mid \hat{\theta}) - D_{\theta'} \nabla_\theta Q(\hat{\theta} \mid \hat{\theta}) \\
\hat{i}_X &= -\left(I - D_\theta \Phi(\hat{\theta})^T \right) D_\theta^2 Q(\hat{\theta} \mid \hat{\theta})
\end{aligned}$$

where

$$\Phi\left(\theta'\right) = \arg\max_\theta Q\left(\theta \mid \theta'\right)$$

is the EM-map.

Implementing the formulas in R

Using the `numDeriv` package.

```r
## Calculate using definition
fisher_def <- function(est, loglik, x)
    hessian(function(par) loglik(par, x = x), x = est)

## Calculate with the three formulas
fisher1 <- function(est, Q)
    -jacobian(function(par) grad(Q, par, par_prime = par), est)

D2Q <- function(est, Q)
    hessian(Q, est, par_prime = est)

fisher2 <- function(est, Q)
    -D2Q(est, Q) -
        jacobian(function(par_prime) grad(Q, est, par_prime = par_prime),
                 est)

fisher3 <- function(est, Q, update_estimates = NULL)
{
    if (is.null(update_estimates))
        update_estimates <- get_update_estimates_general(Q)
    -(diag(length(est)) - t(jacobian(update_estimates, est))) %*%
        D2Q(est, Q)
}
```

They all give approximately the same result

$$\texttt{fisher\_def}: \quad \begin{pmatrix} 57.48859232 & -0.01589346 \\ -0.01589346 & 1.79719876 \end{pmatrix}$$

$$\texttt{fisher1}: \quad \begin{pmatrix} 57.48859187 & -0.01589217 \\ -0.01589422 & 1.79720353 \end{pmatrix}$$

$$\texttt{fisher2}: \quad \begin{pmatrix} 57.4885922 & -0.01589327 \\ -0.0158944 & 1.79719618 \end{pmatrix}$$

$$\texttt{fisher3}: \quad \begin{pmatrix} 57.48859235 & -0.01589349 \\ -0.01589349 & 1.79719876 \end{pmatrix}$$

Empirical Fisher information

$$\hat{\mathcal{I}} = \sum_{i=1}^{n} \nabla_\theta Q_i(\hat{\theta} \mid \hat{\theta}) \nabla_\theta Q_i(\hat{\theta} \mid \hat{\theta})^T.$$

```r
emp_fisher <- function(mu, sigma2, x, nu)
{
    w_means <- (nu + 1) * 0.5 / (0.5 * (1 + ((x - mu)^2 / (sigma2 * nu))))
    dQ_dmu      <- w_means * ((x - mu) / (nu * sigma2))
    dQ_dsigma2 <- -1/(2*sigma2) +
        (w_means * (x - mu)^2) / (2 * nu * sigma2^2 )
    grad_list     <- cbind(mu = dQ_dmu, sigma2 = dQ_dsigma2)
    crossprod(grad_list)
}
```

$$\begin{pmatrix} 57.2396144 & -0.3093977 \\ -0.3093977 & 1.9327784 \end{pmatrix}$$

- Close, but not identical, to the observed Fisher information.