EM algorithm

Maximum Likelihood Estimation of the t-distribution

Lucas Støjko Andersen

University of Copenhagen

October 26, 2022

The t-distribution

The density of the non-standard t-distribution

x The density of the t-distribution is given by

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

with parameters $\mu \in \mathbf{R}$, $\nu > 0$, $\sigma^2 > 0$. Here, Γ is the gamma function.

For independent observations $X_1, X_2, \ldots, X_n$ with density as in (1) there exist no nice analytic solutions to the MLE — even with ν known.

There is another way

A joint approach with latent variables

Consider (X, W) with density

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

Notice that $W \mid X = x \sim \Gamma(\alpha, \beta)$ — the gamma distribution — with parameters

$$\alpha = \frac{\nu+1}{2} \qquad \beta = \frac{1}{2}\left(1 + \frac{(x-\mu)^2}{\nu\sigma^2}\right). \tag{3}$$

This can be used to show that X indeed has the marginal density of the t-distribution with parameters μ, σ^2, ν.

The Full Maximum Likelihood Estimate

MLE with fixed ν

Assume $\nu > 0$ known. I.i.d. $(X_1, W_1), (X_2, W_2), \ldots, (X_n, W_n)$ have log-likelihood

$$\ell(\theta) \simeq -\frac{n}{2}\log\sigma^2 - \frac{1}{2\nu\sigma^2}\sum_{i=1}^{n} W_i(X_i - \mu)^2. \tag{4}$$

The solution is that of the weighted least squares:

$$\hat{\mu} = \frac{\sum_{i=1}^{n} W_i X_i}{\sum_{i=1}^{n} W_i}, \qquad \hat{\sigma}^2 = \frac{1}{n\nu}\sum_{i=1}^{n} W_i(X_i - \hat{\mu})^2. \tag{5}$$

The Full Maximum Likelihood Estimate

MLE with fixed ν — implemented in R

We can sample (X, W) by sampling W from a χ^2_ν distribution and then sampling X from a $\mathcal{N}(\mu, \nu\sigma^2/W)$.

```
simulate_X <- function(N, mu, sigma, nu) {
    W <- rchisq(N, nu)
    X <- rnorm(N, mu, sqrt(nu * sigma / W))

    list(x = X, w = W)
}

full_mle <- function(X, W, nu) {
    mu <- sum(X * W) / sum(W)
    sigma <- sum(W * (X - mu)^2) / (nu * (length(X)))
    list(mu = mu, sigma = sigma)
}

set.seed(3939392)
samples <- simulate_X(100000, 5, 1.5, 3)
full_mle(samples$x, samples$w, 3)
```

We obtain the estimates $\hat{\mu}_{\text{full}} = 4.997852$ and $\hat{\sigma}^2_{\text{full}} = 1.500886$.

EM algorithm

MLE of the marginal likelihood for fixed ν

Only X is observed and so the full MLE cannot be computed. Using the EM algorithm we iteratively maximize the quantity

$$Q(\theta \mid \theta') = E_{\theta'}(\log f_\theta(X, W) \mid X) \tag{6}$$

where $\theta = (\mu, \sigma^2)$. Using the log-likelihood from earlier

$$Q(\theta \mid \theta') \simeq -\frac{n}{2}\log\sigma^2 - \frac{1}{2\nu\sigma^2}\sum_{i=1}^{n} E_{\theta'}(W_i \mid X_i)(X_i - \mu)^2. \tag{7}$$

The maximizer is then

$$\hat{\mu}_{\theta'} = \frac{\sum_{i=1}^{n} E_{\theta'}(W_i \mid X_i)X_i}{\sum_{i=1}^{n} E_{\theta'}(W_i \mid X_i)}, \quad \hat{\sigma}^2_{\theta'} = \frac{1}{n\nu}\sum_{i=1}^{n} E_{\theta'}(W_i \mid X_i)(X_i - \hat{\mu})^2.$$

EM algorithm
E-step and M-step

The E-step boils down to computing $E_{\theta'}(W_i \mid X_i)$

$$E_{\theta'}(W_i \mid X_i) = \frac{\alpha}{\beta} = \frac{\nu' + 1}{2} \frac{1}{\frac{1}{2}\left(1 + \frac{(X_i - \mu')^2}{\nu' \sigma'^2}\right)} = \frac{\nu' + 1}{1 + \frac{(X_i - \mu')^2}{\nu' \sigma'^2}} \tag{8}$$

and doing the M-step by computing

$$\hat{\mu}_{\theta'} = \frac{\sum_{i=1}^n E_{\theta'}(W_i \mid X_i) X_i}{\sum_{i=1}^n E_{\theta'}(W_i \mid X_i)}, \quad \hat{\sigma}^2_{\theta'} = \frac{1}{n\nu} \sum_{i=1}^n E_{\theta'}(W_i \mid X_i)(X_i - \hat{\mu})^2.$$

EM algorithm

Implementation of marginal MLE

```
E_step <- function(x, nu) {
    #force(x)
    #force(nu)
    function(par) {
        # mu = par[1]
        # sigma^2 = par[2]
        (nu + 1) / (1 + ((x - par[1])^2) / (nu * par[2]))
    }
}

M_step <- function(x, nu) {
  #force(x)
  #force(nu)
  function(EW) {
    mu <- sum(EW * x) / sum(EW)
    sigma <- mean(EW * (x - mu)^2) / nu
    c(mu, sigma)
  }
}
```

EM algorithm

Implementation of marginal MLE

```
EM <- function(par, x, nu, maxit = 500, min.eps = 1e-7) {
    E <- E_step(x, nu)
    M <- M_step(x, nu)
    for(i in 1:maxit) {
        EW <- E(par)
        new_par <- M(EW)
        if(sum((new_par - par)^2) < min.eps * (sum(par^2) + min.eps)) {
            par <- new_par
            break
        }
        par <- new_par
        if(i == maxit) warning("Maximum number of itertaions reached.")
    }
    names(par) <- c("mu", "sigma")
    list(par = c(par, nu = nu), iterations = i)
}
```

EM algorithm

Comparison of marginal MLE and full MLE

For starting values $\theta' = (1, 2)$ we obtain the estimates

$$\hat{\mu}_{\mathsf{EM}} = 4.995958, \qquad \hat{\sigma}^2_{\mathsf{EM}} = 1.504293, \qquad \nu = 3. \tag{9}$$

in 9 iterations of the EM algorithm. Recall the full MLE

$$\hat{\mu}_{\mathsf{full}} = 4.997852, \qquad \hat{\sigma}^2_{\mathsf{full}} = 1.500886, \qquad \nu = 3. \tag{10}$$

Robustness of the initial parameters

Properties of the t-distribution

The t-distribution has first moment if $\nu > 1$ and second moment if $\nu > 2$. When the moments exist then

$$EX = \mu, \qquad VX = \sigma^2 \frac{\nu}{\nu + 2}. \tag{11}$$

The calculations in the E-step may be unstable

$$\hat{\mu}_{\theta'} = \frac{\frac{1}{n}\sum_{i=1}^{n} E_{\theta'}(W_i \mid X_i)X_i}{\frac{1}{n}\sum_{i=1}^{n} E_{\theta'}(W_i \mid X_i)}, \quad \hat{\sigma}^2_{\theta'} = \frac{1}{n\nu}\sum_{i=1}^{n} E_{\theta'}(W_i \mid X_i)(X_i - \hat{\mu})^2.$$

Robustness of the intial parameters

Gaussian resemblence of the t-distribution for large ν

When $\nu \longrightarrow \infty$ the density of the t-distribution approaches the density of a normal distribution with mean μ and variance σ^2. It does so very quickly!

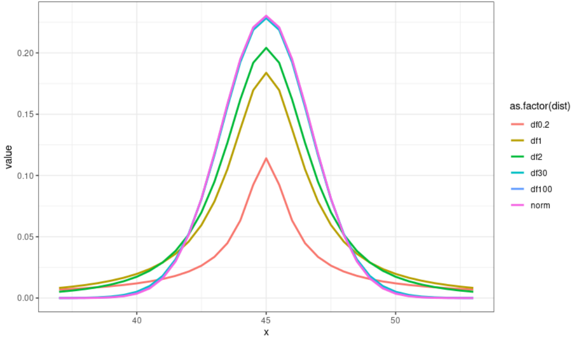


Figure: $\mu = 45$, $\sigma^2 = 3$.

Robustness of the intial parameters

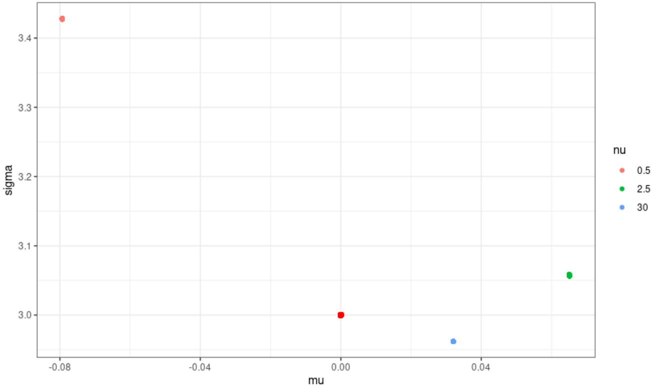
Testing random intial parameters

```
test_robustness <- function(mu, sigma, nu, n) {
    result <- vector("list", n)
    initial_par <- numeric(2 * n)
    dim(initial_par) <- c(n, 2)
    X <- extraDistr::rlst(2000, df = nu, mu = mu, sigma = sqrt(sigma))
    for(i in 1:n) {
        mu_r <- rcauchy(1, location = mu, scale = 100)
        sigma_r <- extraDistr::rpareto(1, a = 0.15, b = sigma)
        initial_par[i, 1] <- mu_r
        initial_par[i, 2] <- sigma_r
        result[[i]] <- EM(c(mu_r, sigma_r), X, nu)
    }
    list(results = result, initial_par = initial_par)
}
```

Robustness of the intial parameters

Testing random intial parameters

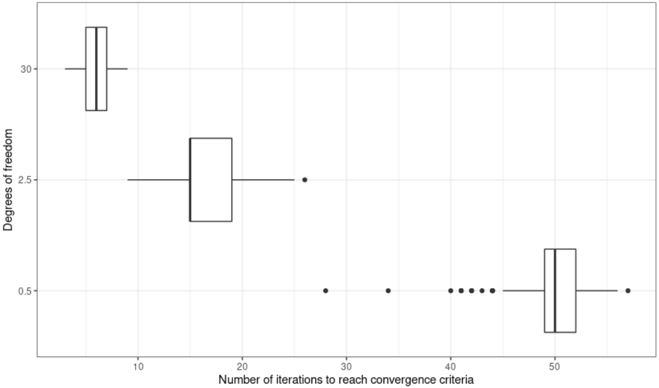
Testing initial parameters with $\mu = 0$, $\sigma^2 = 3$.



Robustness of the intial parameters

Testing random intial parameters

Testing initial parameters with $\mu = 0$, $\sigma^2 = 3$.



Automatic inital parameters

Median and inter quantile range

Sample mean and sample variance are unstable for $\nu < 2$. Suggestion: use inital $\theta_0 = (\text{median}(X), \text{IQR}(X))$.

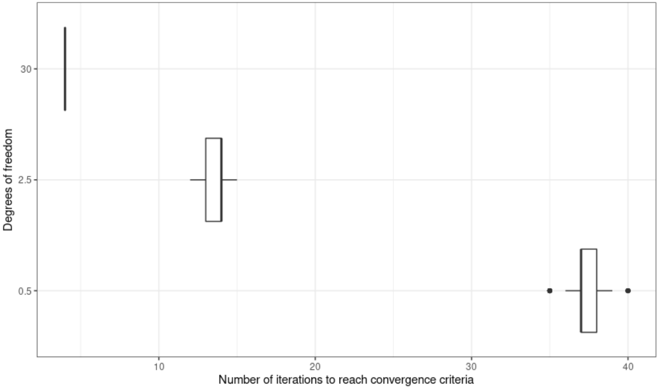
```
EM <- function(par = NULL, x, nu, maxit = 500, min.eps = 1e-7) {
    E <- E_step(x, nu)
    M <- M_step(x, nu)
    if(is.null(par)) {
        par <- c(median(x), IQR(x))
    }
    .
    .
    .
test_robustness2 <- function(mu, sigma, nu, n) {
    result <- vector("list", n)

    for(i in 1:n) {
        X <- extraDistr::rlst(2000, df = nu, mu = mu, sigma = sqrt(sigma))
        result[[i]] <- EM(x = X, nu = nu)
    }
    result
}
```

Automatic inital parameters

Testing of automatic parameters

Using $\mu = 0$ and $\sigma = 3$.



Newton-Methods

MLE directly from the marginal log-likelihood

Instead of using the EM algorithm to maximize the marginal likelihood one could maximize the likelihood directly:

$$\ell(\theta) \simeq -\frac{n}{2}\log\sigma^2 - \frac{\nu+1}{2}\sum_{i=1}^{n}\log\left(1+\frac{(X_i-\mu)^2}{\nu\sigma^2}\right). \tag{12}$$

We have the gradient

$$\frac{\partial}{\partial\mu}\ell(\theta) = \frac{\nu+1}{\nu\sigma^2}\sum_{i=1}^{n}\frac{X_i-\mu}{1+\frac{(X_i-\mu)^2}{\nu\sigma^2}}$$

$$\frac{\partial}{\partial\sigma^2}\ell(\theta) = -\frac{n}{2\sigma^2} + \frac{\nu+1}{2\nu(\sigma^2)^2}\sum_{i=1}^{n}\frac{(X_i-\mu)^2}{1+\frac{(X_i-\mu)^2}{\nu\sigma^2}}$$

First Order Methods

Direct optimization of the marignal likelihood

Implemented minimizing the negative average log likelihood

$$-\frac{1}{n}\sum_{i=1}^{n}\ell_i(\theta). \tag{13}$$

```
logl <- function(x, nu) {
    n <- length(x)
    force(nu)

    function(par) {
        mu <- par[1]
        sigma <- par[2]

        K <- sum(log(1 + (x - mu)^2 / (nu * sigma)))

        log(sigma) / 2 + (nu + 1) * K / (2 * n)
    }
}
```

First Order Methods

Direct optimization of the marignal likelihood

```
gradl <- function(x, nu) {
    n <- length(x)
    force(nu)

    function(par) {
        mu <- par[1]
        sigma <- par[2]

        C1 <- (x - mu) / (1 + (x - mu)^2 / (nu * sigma))

        K_mu <- sum(C1)
        K_sigma <- sum(C1 * (x - mu))

        grad_mu <- -(nu + 1) * K_mu / (n * nu * sigma)
        grad_sigma <- 1 / (2 * sigma) - (nu + 1) * K_sigma / (2 * n * nu * sigma^2)

        c(grad_mu, grad_sigma)
    }
}
```

First Order Methods

Implementation of Gradient Descent with backtracking

```
GD <- function(par, H, gr, d = 0.8, c = 0.1, gamma0 = 1, epsilon = 1e-7, maxiter = 500, cb = NULL
) {
    for(i in 1:maxiter) {
    value <- H(par)
    grad <- gr(par)
    h_prime <- sum(grad^2)
    gamma <- gamma0
    par1 <- par - gamma * grad
    if(!is.null(cb)) cb()
    while(min(H(par1), Inf, na.rm = TRUE) > value - c * gamma * h_prime) {
        gamma <- d * gamma
        par1 <- par - gamma * grad
    }
    if(norm(par - par1, "2") < epsilon * (norm(par, "2") + epsilon)) break
    par <- par1
    }
    if(i == maxiter) warning("Maximal number, ", maxiter, ", of iterations reached")
    par1
}
```

First Order Methods

Implementation of the Conjugate Gradient Descent algorithm

```
CG <- function(par, H, gr, d = 0.8, c = 0.1, gamma0 = 1, epsilon = 1e-7, maxiter = 500, cb = NULL) {
    p <- length(par)
    m <- 1
    rho0 <- numeric(p)
    for(i in 1:maxiter) {
    value <- H(par)
    grad <- gr(par)
    grad_norm_sq <- sum(grad^2)
    if(!is.null(cb)) cb()
    gamma <- gamma0
    rho <- - grad + grad_norm_sq * rho0
    h_prime <- drop(t(grad) %*% rho)
    if(m > p || h_prime >= 0) {
        rho <- - grad
        h_prime <- - grad_norm_sq
        m <- 1
    }
    par1 <- par + gamma * rho
    while(min(H(par1), Inf, na.rm = TRUE) > value + c * gamma * h_prime) {
        gamma <- d * gamma
        par1 <- par + gamma * rho
    }
    rho0 <- rho / grad_norm_sq
    if(norm(par - par1, "2") < epsilon * (norm(par, "2") + epsilon)) break
    par <- par1
    m <- m + 1
    }
    if(i == maxiter) warning("Maximal number, ", maxiter, ", of iterations reached")
    par1
}
```

Second Order Methods

Implementation of the Hessian

```
hessl <- function(x, nu) {
    n <- length(x)
    force(nu)
    function(par){
        mu <- par[1]
        sigma <- par[2]

        C0 <- 1 / (1 + (x - mu)^2 / (nu * sigma))
        C1 <- C0 * (x - mu)
        C2 <- C1 * (x - mu)

        hess_mu <- (nu + 1) * sum(C0) / (n * nu * sigma) +
            2 * (nu + 1) * sum(C1^2) / (n * (nu * sigma)^2)

        hess_sigma <- -1 / (2 * sigma^2) +
            (nu + 1) * sum(C2) / (n * nu * sigma^3) -
            (nu + 1) * sum(C2^2) / (2 * n * nu^2 * sigma^4)

        hess_mu_sigma <- (nu + 1) * sum(C1) / (n * nu * sigma^2) -
            (nu + 1) * sum(C1^2 * (x - mu)) / (n * nu * sigma^3)

        hess <- c(hess_mu, hess_mu_sigma, hess_mu_sigma, hess_sigma)
        dim(hess) <- c(2, 2)
        hess
    }
}
```

Second Order Methods

Implementation of Newton algorithm with backtracking and Wolff line search

```
Newton <- function(par, H, gr, hess, d = 0.8, c = 0.2, gamma0 = 1, epsilon = 1e-7, maxiter = 500, cb = NULL) {
    for(i in 1:maxiter) {
        value <- H(par)
        grad <- gr(par)
        if(!is.null(cb)) cb()
        Hessian <- hess(par)
        rho <- - drop(solve(Hessian, grad))
        gamma <- gamma0
        par1 <- par + gamma * rho
        h_prime <- t(grad) %*% rho
        while(min(H(par1), Inf, na.rm = TRUE) > value +  c * gamma * h_prime) {
            gamma <- d * gamma
            par1 <- par + gamma * rho
        }
        if(norm(par - par1, "2") < epsilon * (norm(par, "2") + epsilon)) break
        par <- par1
    }
    if(i == maxiter) warning("Maximal number, ", maxiter, ", of iterations reached")
    par1
}
```

Comparisons of algorithms

Comparison of convergence

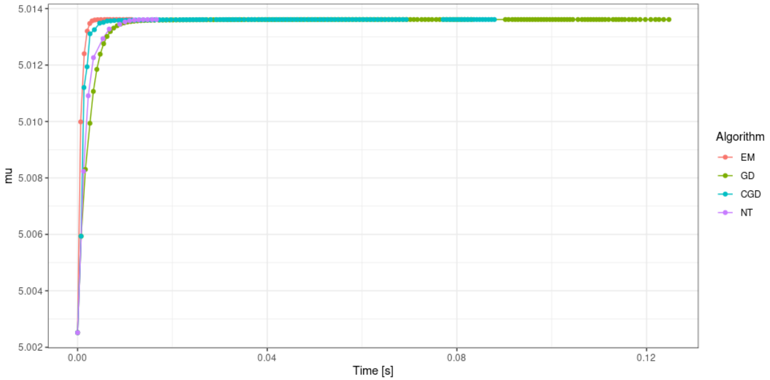
The convergence criteria is identical for both algorithms. When either the algorithms reach 500 iterations or $||\theta_{n+1} - \theta_n|| < \varepsilon(||\theta_n|| + \varepsilon)$ for $\varepsilon = 10^{-7}$.

10.000 samples of the t-distribution with parameters $\mu = 5$, $\sigma^2 = 2$ and $\nu = 3$. For the GD, CGD and Newton algorithms the paremters $\mathtt{d} = 0.8$, $\mathtt{c} = 0.2$ and $\mathtt{gamma0} = 1$ were used. All algorithms reached convergence before 500 iterations.

$$
\begin{aligned}
\hat{\theta}_{\text{GD}} &= (5.013613671, 2.051342021) & \ell(\hat{\theta}_{\text{GD}}) &= 1.12761474375058723 \\
\hat{\theta}_{\text{CG}} &= (5.013613671, 2.051342056) & \ell(\hat{\theta}_{\text{CG}}) &= 1.12761474375060544 \\
\hat{\theta}_{\text{NT}} &= (5.013613344, 2.051333484) & \ell(\hat{\theta}_{\text{NT}}) &= 1.12761474374844051 \\
\hat{\theta}_{\text{EM}} &= (5.013613675, 2.051333775) & \ell(\hat{\theta}_{\text{EM}}) &= 1.12761474374842496
\end{aligned}
$$

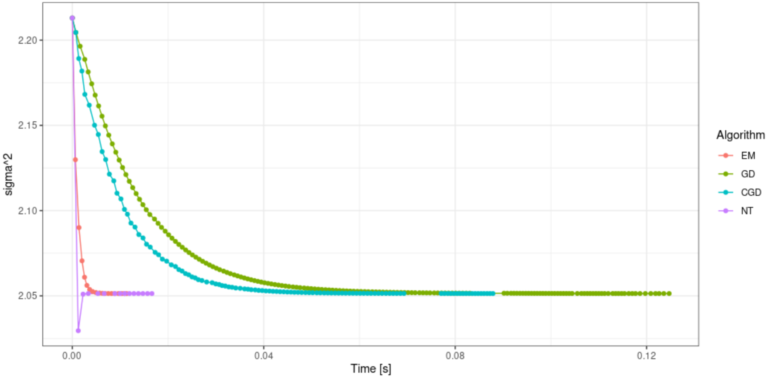
Comparisons of algorithms

Development of the μ-estimate



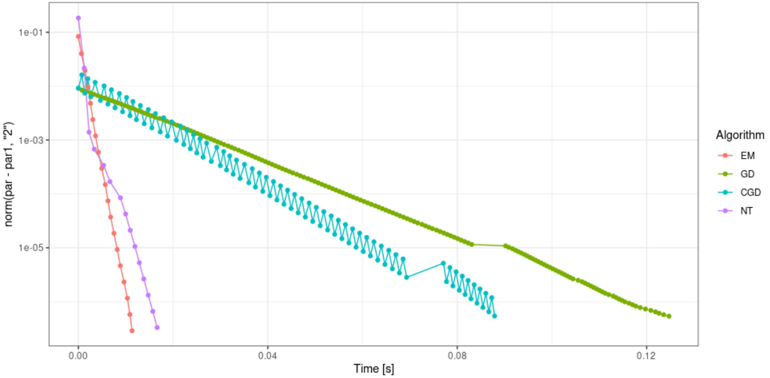
Comparisons of algorithms

Development of the σ^2-estimate



Comparisons of algorithms

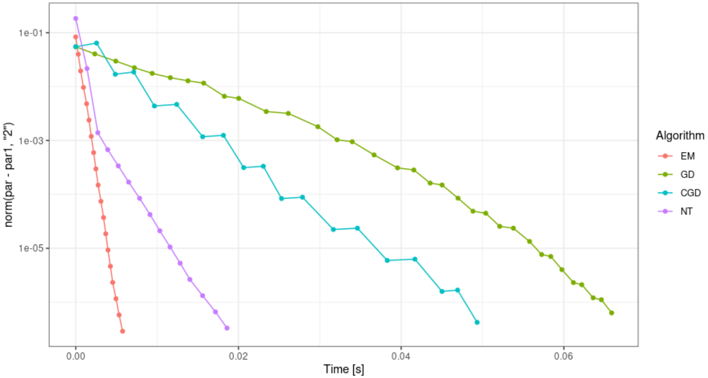
Development of the distance between parameters



Comparisons of algorithms

Parameters matter

Choosing `gamma0` $= 6$ for the GD and CGD algorithms yields



Comparison of algorithms

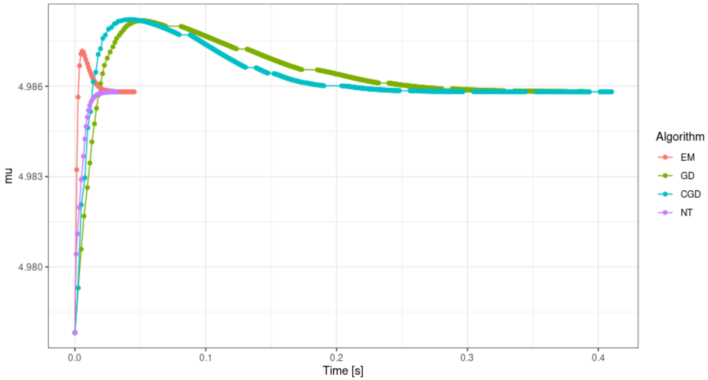
Convergence for small ν

Repeating the same experiment with $\nu = 0.5$. The GD and CGD algorithms did not reach convergence before 500 iterations.

$$
\begin{aligned}
\hat{\theta}_{\mathrm{GD}} &= (4.985824153, 2.107854186) & \ell(\hat{\theta}_{\mathrm{GD}}) &= 2.75212300025436729 \\
\hat{\theta}_{\mathrm{CG}} &= (4.985815963, 2.103884595) & \ell(\hat{\theta}_{\mathrm{CG}}) &= 2.75212286766256398 \\
\hat{\theta}_{\mathrm{NT}} &= (4.985814613, 2.103818326) & \ell(\hat{\theta}_{\mathrm{NT}}) &= 2.75212286762692315 \\
\hat{\theta}_{\mathrm{EM}} &= (4.985815835, 2.103821265) & \ell(\hat{\theta}_{\mathrm{EM}}) &= 2.75212286762684455
\end{aligned}
$$

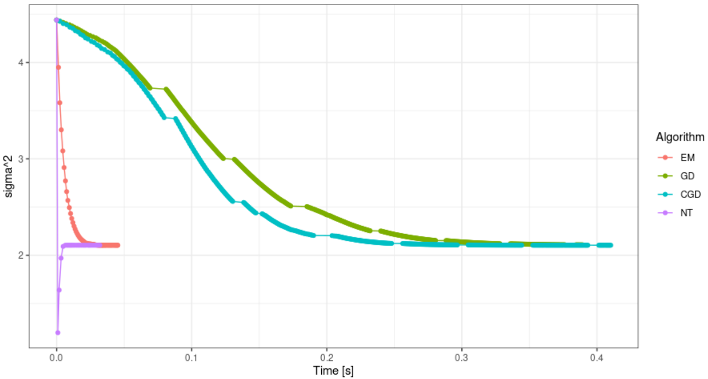
Comparisons of algorithms

Development of the μ-estimate



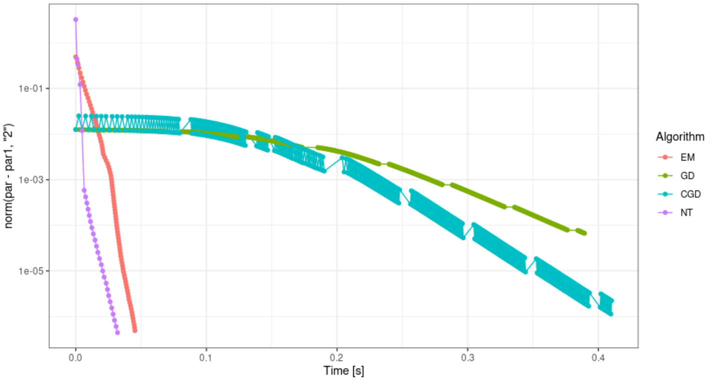
Comparisons of algorithms

Development of the σ^2-estimate



Comparisons of algorithms

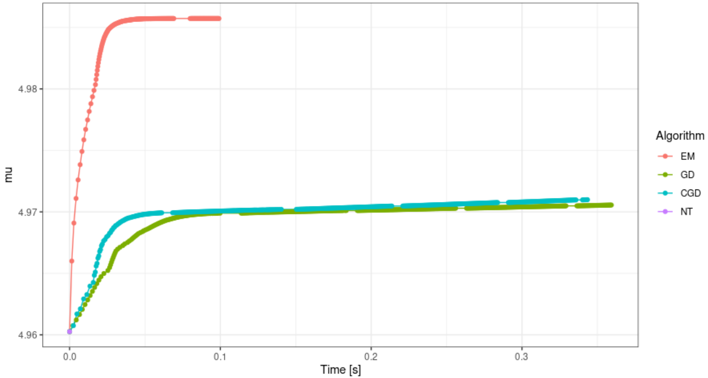
Development of the distance between parameters



Comparisons of algorithms

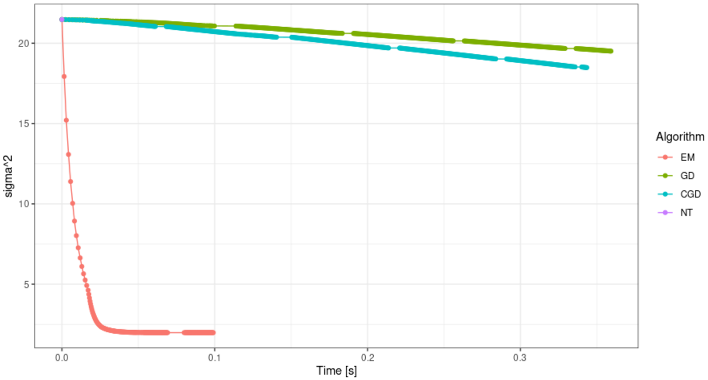
Stability of algorithms

If $\nu = 0.2$ then the Newton algorithm no longer works. It gets stuck in backtracking. The initial guess $(\text{median}(X), \text{IQR}(X))$ is too far off — in particular the guess of σ^2 is not very good.



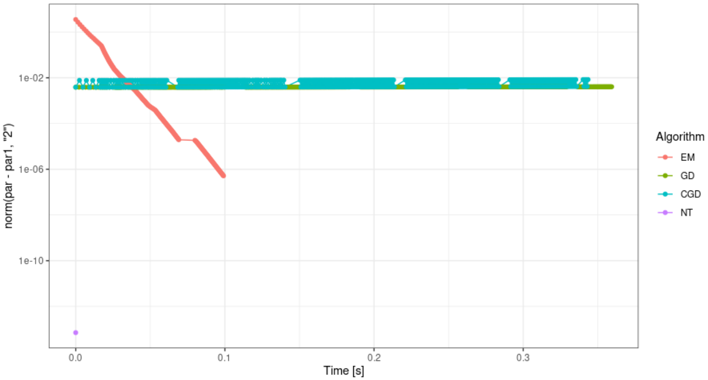
Comparisons of algorithms

Stability of algorithms



Comparisons of algorithms

Stability of algorithms



Profiling of the Newton algorithm

Using 10.000.000 samples with $\mu = 5$, $\sigma^2 = 2$ and $\nu = 1.5$.

Newton.R

```r
Newton <- function(par, H, gr, hess, d = 0.8, c = 0.2, gamma0 = 1, epsilon = 1e-7,
    maxiter = 500, cb = NULL) {
  for(i in 1:maxiter) {
    value <- H(par)
    grad <- gr(par)
    Hessian <- hess(par)
    rho <- - drop(solve(Hessian, grad))
    gamma <- gamma0
    par1 <- par + gamma * rho
    h_prime <- t(grad) %*% rho
    while(min(H(par1), Inf, na.rm = TRUE) > value +  c * gamma * h_prime) {
      gamma <- d * gamma
      par1 <- par + gamma * rho
    }
    if(!is.null(cb)) cb()
    if(norm(par - par1, "2") < epsilon * (norm(par, "2") + epsilon)) break
    par <- par1
  }
  if(i == maxiter)
    warning("Maximal number, ", maxiter, ", of iterations reached")
  par1
}
```

Line	Code	Memory (freed)	Memory (allocated)	Time
3	`value <- H(par)`		991.9	2565
4	`grad <- gr(par)`		2975.5	3165
5	`Hessian <- hess(par)`	-6408.8	6484.7	6270
6	`rho <- - drop(solve(Hessian, grad))`			5
10	`while(min(H(par1), Inf, na.rm = TRUE) > value +  c * gamma * h_prime) {`	-4653.9	686.6	3300
15	`if(norm(par - par1, "2") < epsilon * (norm(par, "2") + epsilon)) break`			10

Profiling of the EM algorithm

Using 10.000.000 samples with $\mu = 5$, $\sigma^2 = 2$ and $\nu = 1.5$.

EM-algorithm.R

```r
E_step <- function(x, nu) {
  function(par) {
    mu <- par[1]
    sigma <- par[2]
    test <- (nu + 1) / (1 + ((x - mu)^2) / (nu * sigma))
    test
  }
}

M_step <- function(x, nu) {
  function(EW) {
    mu <- sum(EW * x) / sum(EW)
    sigma <- mean(EW * (x - mu)^2) / nu
    c(mu, sigma)
  }
}

EM <- function(par = NULL, x, nu, cb = NULL, maxit = 500, min.eps = 1e-7) {
  E <- E_step(x, nu)
  M <- M_step(x, nu)
  if(is.null(par)) {
    par <- c(median(x), IQR(x))
  }
  for(i in 1:maxit) {
    EW <- E(par)
    par1 <- M(EW)
    if(!is.null(cb)) cb()
    if(norm(par - par1, "2") < min.eps * (norm(par, "2") + min.eps)) break
    par <- par1
  }
  if(i == maxit) warning("Maximum number of itertaions ", maxit, " reached.")
  names(par1) <- c("mu", "sigma")
  list(par1 = c(par, nu = nu), iterations = i)
}
```

Code	Memory		Time
`test <- (nu + 1) / (1 + ((x - mu)^2) / (nu * sigma))`	-1831.1	2060.1	5310
`mu <- sum(EW * x) / sum(EW)`	-1831.1	1907.3	2680
`sigma <- mean(EW * (x - mu)^2) / nu`	-3662.7	3814.7	2760
`M <- M_step(x, nu)`			10
`EW <- E(par)`	-1831.1	2060.1	5310
`par1 <- M(EW)`	-3662.4	3814.7	5500

Calculating the fisher information

Methods of calculation the fisher information