A Note on the Newton-Raphson Algorithm

Econometrics, Winter 2021

Prepared by Micole De Vera

Suppose we want to minimize a function $Q(\boldsymbol{\beta})$, where $\boldsymbol{\beta}$ is a $k \times 1$ vector and $Q(\cdot)$ is assumed to be twice-continuously differentiable. We are interested in the minimizer $\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} Q(\boldsymbol{\beta})$.

Let

- $\mathbf{g}(\boldsymbol{\beta}) = \dfrac{\partial Q(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}$ be the gradient of $Q(\boldsymbol{\beta})$ which is a $k \times 1$ vector with typical element $\dfrac{\partial Q(\boldsymbol{\beta})}{\partial \beta_i}$
- $\mathbf{H}(\boldsymbol{\beta}) = \dfrac{\partial^2 Q(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}'}$ be the Hessian of $Q(\boldsymbol{\beta})$ which is a $k \times k$ matrix with typical element $\dfrac{\partial^2 Q(\boldsymbol{\beta})}{\partial \beta_i \partial \beta_j}$

Motivation

In econometric applications, we usually think of the function $Q(\cdot)$ as an estimation criterion that takes in a parameter vector $\boldsymbol{\beta}$. Then $\widehat{\boldsymbol{\beta}}$ is taken as an estimator of the unknown true parameter $\boldsymbol{\beta}_0$. Many estimators can be framed as minimization problems:

(i) Given a random sample $\{(y_i, \mathbf{x}_i')\}_{i=1}^n$, the OLS estimator for the coefficients of a linear regression of y on x minimizes the least squares criterion

$$Q(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{x}_i'\boldsymbol{\beta})^2 = \frac{(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{2}$$

with gradient and Hessian

$$\mathbf{g}(\boldsymbol{\beta}) = -\sum_{i=1}^{n}\mathbf{x}_i(y_i - \mathbf{x}_i'\boldsymbol{\beta}) = -\mathbf{X}'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

$$\mathbf{H}(\boldsymbol{\beta}) = \sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i' = \mathbf{X}'\mathbf{X}$$

(ii) Given a random sample $\{(y_i, \mathbf{x}_i')\}_{i=1}^n$, the nonlinear least squares (NLS) estimator corresponding to the nonlinear regression model:

$$y_i = f(\mathbf{x}_i, \boldsymbol{\beta}) + \varepsilon_i$$
$$\mathbb{E}[\varepsilon_i | X] = 0$$

minimizes

$$Q(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i, \beta))^2$$

(iii) We have a model for $\mathbf{z}$ such that it has density function $f(\mathbf{z}|\boldsymbol{\beta})$. We get a random sample $\{\mathbf{z}_i\}_{i=1}^n$ and we are interested in estimating the unknown parameter $\boldsymbol{\beta_0}$. The MLE of the parameter minimizes

$$Q(\boldsymbol{\beta}) = -\sum_{i=1}^{n}\log f(\mathbf{z}_i|\boldsymbol{\beta})$$

with gradient and Hessian

$$\mathbf{g}(\boldsymbol{\beta}) = -\sum_{i=1}^{n}\frac{\partial \log f(\mathbf{z}_i|\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}$$

$$\mathbf{H}(\boldsymbol{\beta}) = -\sum_{i=1}^{n}\frac{\partial^2 \log f(\mathbf{z}_i|\boldsymbol{\beta})}{\partial \boldsymbol{\beta}\partial \boldsymbol{\beta}'}$$

Newton-Raphson Algorithm

Consider a second-order Taylor approximation $Q^*(\boldsymbol{\beta})$ to the function $Q(\boldsymbol{\beta})$ around an initial value $\boldsymbol{\beta}^{(0)}$:

$$Q^*(\boldsymbol{\beta}) = Q(\boldsymbol{\beta}^{(0)}) + \mathbf{g}(\boldsymbol{\beta}^{(0)})^\top(\boldsymbol{\beta} - \boldsymbol{\beta}^{(0)}) + \frac{1}{2}(\boldsymbol{\beta} - \boldsymbol{\beta}^{(0)})^\top\mathbf{H}(\boldsymbol{\beta}^{(0)})(\boldsymbol{\beta} - \boldsymbol{\beta}^{(0)})$$

A minimizer $\boldsymbol{\beta}^{(1)}$ of $Q^*(\cdot)$ will satisfy the following necessary first-order conditions with respect to $\boldsymbol{\beta}$

$$\mathbf{g}(\boldsymbol{\beta}^{(0)}) + \mathbf{H}(\boldsymbol{\beta}^{(0)})(\boldsymbol{\beta}^{(1)} - \boldsymbol{\beta}^{(0)}) = 0$$

or

$$\boldsymbol{\beta}^{(1)} = \boldsymbol{\beta}^{(0)} - \mathbf{H}(\boldsymbol{\beta}^{(0)})^{-1}\mathbf{g}(\boldsymbol{\beta}^{(0)}) \ .$$

Note that $\boldsymbol{\beta}^{(1)}$ happens to be the global minimizer of $Q^*(\boldsymbol{\beta})$ when $Q^*(\boldsymbol{\beta})$ is a strictly convex function, that is, if and only if $\mathbf{H}(\boldsymbol{\beta}^{(0)})$ is positive definite.

Note that if the function $Q(\cdot)$ was quadratic, $Q^*(\boldsymbol{\beta})$ would be exact and not an approximation.
Then, $\boldsymbol{\beta}^{(1)}$ coincides with $\widehat{\boldsymbol{\beta}}$ as a minimizer of $Q(\cdot)$. However, when $Q(\cdot)$ is not quadratic but the quadratic approximation $Q^*(\boldsymbol{\beta})$ is a good approximation, then $\boldsymbol{\beta}^{(1)}$ should be close to $\widehat{\boldsymbol{\beta}}$. For problems that minimize sum-of-squares functions (as in ordinary/nonlinear least squares), $Q(\boldsymbol{\beta})$ is approximately quadratic when sufficiently close to their minima.

The Newton-Raphson algorithm involves using this quadratic approximation idea to get a sequence of updates $\boldsymbol{\beta}^{(1)}, \boldsymbol{\beta}^{(2)}, ...$ using

$$\boldsymbol{\beta}^{(m+1)} = \boldsymbol{\beta}^{(m)} - \mathbf{H}(\boldsymbol{\beta}^{(m)})^{-1}\mathbf{g}(\boldsymbol{\beta}^{(m)}) \text{ for } m = 1, 2, ...$$

If the sequence $\{\boldsymbol{\beta}^{(m)}\}_{m=1}^{\infty}$ converges to a limit $\boldsymbol{\beta}^*$ it will be such that

$\boldsymbol{\beta}^* = \boldsymbol{\beta}^* - \mathbf{H}(\boldsymbol{\beta}^*)^{-1}\mathbf{g}(\boldsymbol{\beta}^*)$ or $\mathbf{H}(\boldsymbol{\beta}^*)^{-1}\mathbf{g}(\boldsymbol{\beta}^*) = 0$

which since $\mathbf{H}(\boldsymbol{\beta}^*)^{-1}$ is invertible implies $\mathbf{g}(\boldsymbol{\beta}^*) = 0$. This means that $\boldsymbol{\beta}^*$ is a critical point of $Q(\cdot)$.
If $\mathbf{H}(\boldsymbol{\beta}^*)$ is positive definite, then $\boldsymbol{\beta}^*$ is a local minimizer of $Q(\cdot)$.

Additional Remarks and Computational Concerns

- In practice, we need some criteria to determine when the updating process should stop. That is, we need some criteria to decide whether $\boldsymbol{\beta}^{(m)}$ already provides a sufficiently accurate approximation to $\widehat{\boldsymbol{\beta}}$. Popular stopping criteria include: for some chosen tolerance level $\epsilon > 0$,

1. Change in objective function is small: $|Q(\boldsymbol{\beta}^{(m)}) - Q(\boldsymbol{\beta}^{(m-1)})| < \epsilon$
2. Gradient is close to zero: $||\mathbf{g}(\boldsymbol{\beta}^{(m)})|| < \epsilon$
3. Update in parameters is small: $||\boldsymbol{\beta}^{(m)} - \boldsymbol{\beta}^{(m-1)}|| < \epsilon$

- Note that there are no concrete rules as to which of these stopping rules is the best as they depend on the magnitude of the parameters. They may yield different results if the units of measurement of variables are changed or if the model is reparametrized. Another recommended stopping rule is when $\mathbf{g}(\boldsymbol{\beta}^{(m)})^\top \mathbf{H}(\boldsymbol{\beta}^{(m)})\mathbf{g}(\boldsymbol{\beta}^{(m)}) < \epsilon$. The advantage of this stopping rule is that it weights the various components of the gradient in a manner inversely proportional to the precision with which the corresponding parameters are estimated.
- Any stopping rule will behave badly if the tolerance level ϵ is not chosen well. If ϵ is too large, then the algorithm may stop when it is still too far away from $\widehat{\boldsymbol{\beta}}$. On the other hand, if ϵ is too small, then the algorithm may continue updating even if we have reached a sufficiently good approximation to $\widehat{\boldsymbol{\beta}}$ and any differences result only from rounding errors. In practice, one can experiment different values of ϵ to see how sensitive results are. If the reported $\widehat{\boldsymbol{\beta}}$ changes noticeably when ϵ is reduced, then the initial choice of ϵ may have been too large (or the algorithm is finding it hard to find a minimum).
- It is recommended to also stop the updating process if the covergence criterion is not met after a large number of iterations. When the algorithm terminates in this manner, the convergence is said to fail.
- Where the algorithm starts, $\boldsymbol{\beta}^{(0)}$, may determine how well the algorithm performs (or whether it even converges at all). For functions $Q(\boldsymbol{\beta})$ that are not globally convex, the algorithm may fail when we reach an element in the sequence $\boldsymbol{\beta}^{(j)}$ for which $Q(\boldsymbol{\beta})$ is not convex around the neighborhood of $\boldsymbol{\beta}^{(j)}$. Two pathological cases include:

1. If $Q(\boldsymbol{\beta})$ is concave at $\boldsymbol{\beta}^{(j)}$, this causes the update to head off in the wrong direction and you might end up with a local maximizer instead. Note that the Newton-Raphson algorithm can be interpreted as finding solutions to the system of equations $\mathbf{g}(\boldsymbol{\beta}) = 0$. The solutions to this system are critical points of $Q(\boldsymbol{\beta})$ which may be local minimizers, local maximizers or saddle points.

2. If $Q(\boldsymbol{\beta})$ is quite flat at $\boldsymbol{\beta}^{(j)}$, then the quadratic approximation is very bad for values away from $\boldsymbol{\beta}^{(j)}$. It could be the the update might bring you even further away from the minimizer than where you started and may slow down convergence.

- There are no general and easily verifiable sufficient conditions to guarantee convergence of the sequence $\{\boldsymbol{\beta}^{(m)}\}_{m=1}^{\infty}$ to the global minimizer of an arbitrary function $Q(\boldsymbol{\beta})$. One important exception is when $Q(\boldsymbol{\beta})$ is a strictly convex function in $\boldsymbol{\beta}$ then the global convergence result is guaranteed. A special case of this is when $Q(\boldsymbol{\beta})$ is quadratic with postive definite Hessian -- then, convergence to the global minimizer is guaranteed (in fact, after one iteration). Sometimes, there is a reparametrization $\boldsymbol{\beta} \mapsto \boldsymbol{\lambda}(\boldsymbol{\beta})$, $\boldsymbol{\lambda} : \mathbb{R}^k \to \mathbb{R}^k$ invertible, such that the composite function $Q \circ \boldsymbol{\lambda}^{-1}$ is strictly convex in the parameters $\boldsymbol{\lambda}$ (you will see this in one of the computer exercises).
- In general, $Q(\boldsymbol{\beta})$ may not have only one local minimum. In such cases, the choice of the starting value $\boldsymbol{\beta}^{(0)}$ may determine to which local minimum the algorithm converges to. In practice, when the objective function is suspected not to be globally convex, we try to minimize $Q(\boldsymbol{\beta})$ several times, starting at a number of different starting points (ideally quite dispersed over the interesting regions of the parameter space). If several starting values lead to the same local minimum $\widehat{\boldsymbol{\beta}}$ with $Q(\widehat{\boldsymbol{\beta}})$ less than the value of $Q(\boldsymbol{\beta})$ observed at any other local minimum, then it is plausible (but not certain) that $\widehat{\boldsymbol{\beta}}$ is the global minimizer.
- The Newton-Raphson algorithm converges quickly once it is near a solution but it is computationally expensive for each iteration. This motivated the development of quasi-Newton methods that converge somewhat less rapidly but require much less computational cost (and are often more robust to some of the pathological scenarios like those described above). The idea is that the savings in work per iteration more than offsets the slower convergence. It often is of the form involving a parameter $\alpha^{(j)}$ determined at each iteration, and $\mathbf{B}(\cdot)$ which is an approximation of the Hessian (obtained in a number of ways including neglecting some terms in the true Hessian). The updating equation is given by $\boldsymbol{\beta}^{(m+1)} = \boldsymbol{\beta}^{(m)} - \alpha^{(m)}\mathbf{B}(\boldsymbol{\beta}^{(m)})^{-1}\mathbf{g}(\boldsymbol{\beta}^{(m)})$. For the nonlinear least squares problem which minimizes a sum-of-squares function, one specific quasi-Newton algorithm is the Gauss-Newton regressions which happen to be useful in some cases to obtain standard errors in nonlinear least squares estimation problems.

Example (Unconstrained Optimization of a Multivariate Function)

We will try to find the minimizer of a popular function used to benchmark numerical optimization routines: the Rosenbrock function. Consider the following parametrization

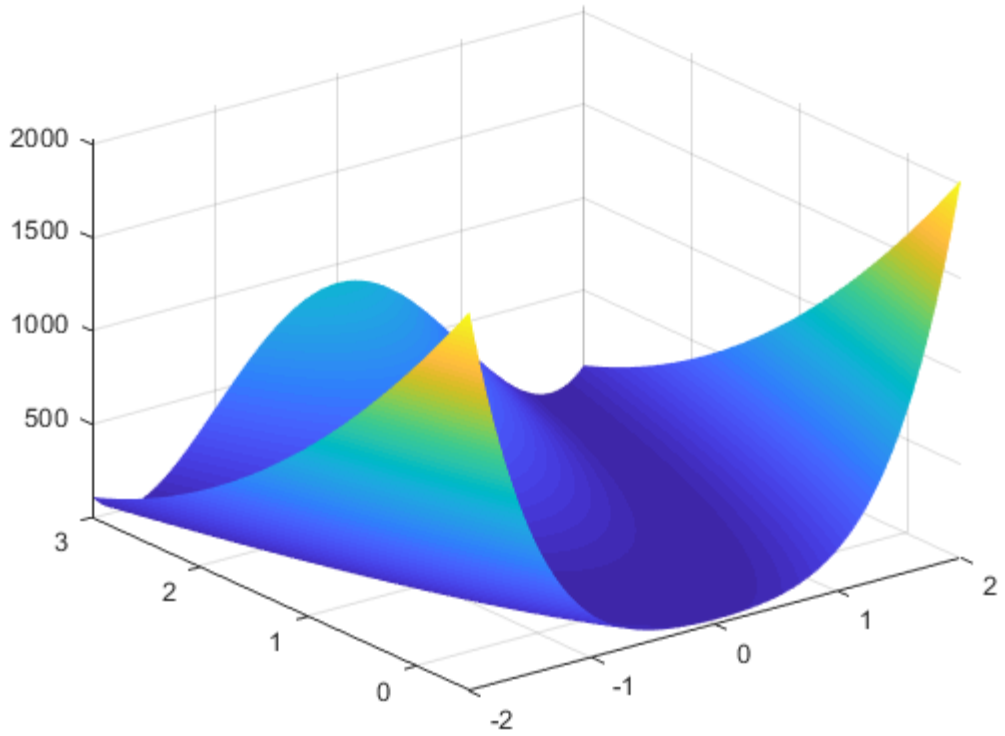
$$f(x, y) = (x - 1)^2 + 100(y - x^2)^2$$

Verify that this function has one global minimum at the point $(1, 1)$.

First, we plot the function to see how it looks like

```
% Define the function
rosenbrock = @(x,y) (x-1).^2 + 100*(y - x.^2).^2;

% Plot the function
x = linspace(-2, 2);
y = linspace(-0.5, 3);
[X, Y] = meshgrid(x, y);
```

```
f = rosenbrock(X, Y);

surf(X, Y, f)
shading interp;
axis tight;
```



The gradient of the function is

$$\mathbf{g}(x,y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2(x-1) - 400x(y-x^2) \\ 200(y-x^2) \end{bmatrix}$$

```
% Define gradient
grad_rosenbrock = @(x) [2*(x(1) - 1) - 400 * x(1) * (x(2) - x(1)^2); ...
                        200 * (x(2) - x(1)^2)];
```

The Hessian of the function is

$$\mathbf{H}(x,y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2 - 400y + 1200x^2 & -400x \\ -400x & 200 \end{bmatrix}$$

```
% Define Hessian
hess_rosenbrock = @(x) [2 - 400 * x(2) + 1200 * x(1)^2, -400 * x(1); ...
```

```
                        -400 * x(1),                    200];
```

We begin with the initial guess of $(-1, 1)'$ and run the algorithm until convergence or maximum iterations of 10, whichever comes first. We will use the convergence criteria that the norm of the gradient is less than 1e-10.

```
% Calibration
maxiter = 10;
tol = 1e-10;

% Initialize
X_path = [-1;1];
F_path = [rosenbrock(X_path(1, 1), X_path(2, 1))];
iter = 2;
grad_norm = 1;

% Newton-Raphson Iterations
while (iter <= maxiter && grad_norm > tol)
    % Newton-Raphson Update
    X_path(:, iter) = X_path(:, iter - 1) - ...
        hess_rosenbrock(X_path(:, iter - 1)) \ grad_rosenbrock(X_path(:, iter - 1));
    F_path(iter) = [rosenbrock(X_path(1, iter), X_path(2, iter))];

    % Update stopping criteria
    grad_norm = norm(grad_rosenbrock(X_path(:, iter)));

    % Update counter
    iter = iter + 1;
end

fprintf("Minimizer found: (%4.2f, %4.2f); Function value: %4.2f; Number of iterations: %i", ...
                X_path(1, end), X_path(2, end), F_path(end), iter - 1);
```

```
Minimizer found: (1.00, 1.00); Function value: 0.00; Number of iterations: 3
```

What if we started with a different initial starting value, say $(0, 1)'$?

```
% Initialize
X_path = [0;1];
F_path = [rosenbrock(X_path(1, 1), X_path(2, 1))];
iter = 2;
grad_norm = 1;

% Newton-Raphson Iterations
while (iter <= maxiter && grad_norm > tol)
    % Newton-Raphson Update
    X_path(:, iter) = X_path(:, iter - 1) - ...
        hess_rosenbrock(X_path(:, iter - 1)) \ grad_rosenbrock(X_path(:, iter - 1));
    F_path(iter) = [rosenbrock(X_path(1, iter), X_path(2, iter))];

    % Update stopping criteria
    grad_norm = norm(grad_rosenbrock(X_path(:, iter)));
```

```
    % Update counter
    iter = iter + 1;
end

fprintf("Minimizer found: (%4.2f, %4.2f); Function value: %4.2f; Number of iterations: %i", ...
                X_path(1, end), X_path(2, end), F_path(end), iter - 1);
```

```
Minimizer found: (1.00, 1.00); Function value: 0.00; Number of iterations: 6
```

We still find the global minimizer but in slightly more update steps. In this simple example where computing the gradient and Hessian is quick, the increased number of update steps may be inconsequential but in more complex problems, this may be substantial. And in some cases, as we discussed above, picking some starting values may even lead to non-convergence of the algorithm.

Example (Maximum Likelihood Estimation)

Let Y be distributed student's t with K degrees of freedom. Let $X = \theta + Y$ for some unknown scalar θ.

(i) Let us simulate $S = 1000$ random samples of X's for $\theta = 0$ and $K = 3$. Set sample size to 200.

```
% Simulation parameters
N = 200;
theta = 0;
K = 3; % degrees of freedom
S = 1000;

rng(1234);

% Simulate X
p_aux = rand(N, S);
Y = tinv(p_aux, K);
X = Y + theta;
```

(ii) Given each random sample $(X_1, X_2, ..., X_N)$, we want to get the maximum likelihood estimate $\hat{\theta}$ by Newton-Raphson when K is known (i.e. $K = 3$).

The density of X with known degrees of freedom K is proportional to the following

$$f(X) \propto \left(1 + \frac{(X-\theta)^2}{K}\right)^{-\frac{K+1}{2}}$$

Then, up to a constant term C, the log-likelihood of the sample is

$$L(\theta|X) = \sum_{i=1}^{N} -\frac{K+1}{2}\log\left(1 + \frac{(X_i-\theta)^2}{K}\right) + C$$

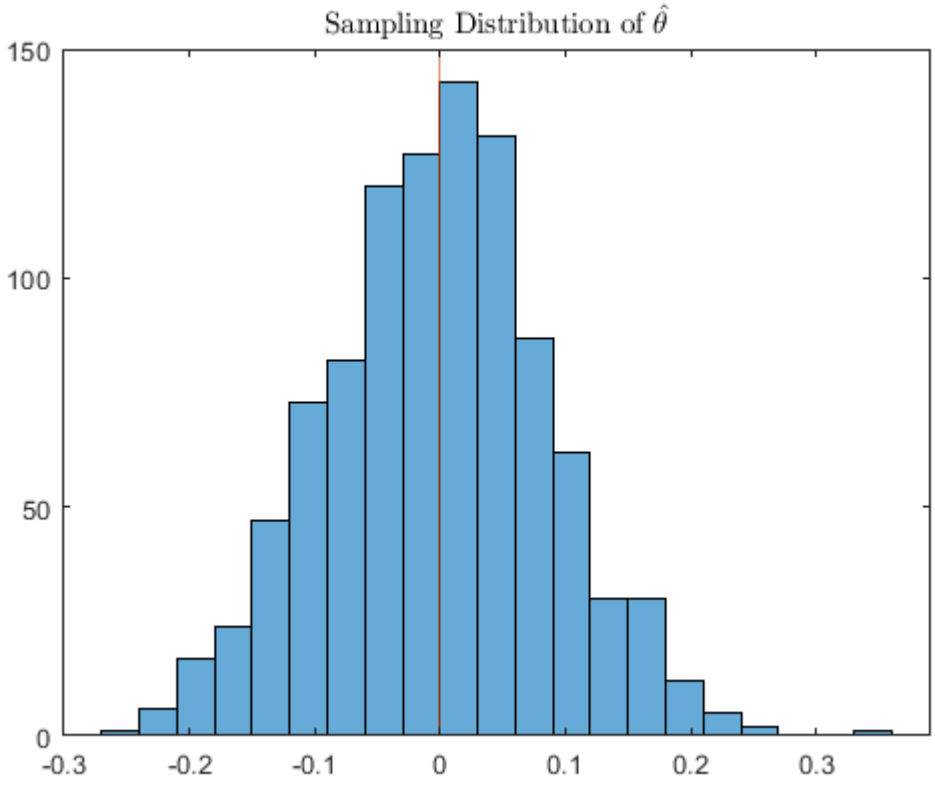
The gradient is

$$\frac{\partial L(\theta|X)}{\partial \theta} = \sum_{i=1}^{N} \frac{K+1}{K} \frac{(X_i - \theta)}{1 + \frac{(X_i - \theta)^2}{K}}$$

The Hessian is

$$\frac{\partial^2 L(\theta|X)}{\partial \theta^2} = \sum_{i=1}^{N} \frac{K+1}{K} \left[\frac{2(X_i - \theta)^2}{K} \left(1 + \frac{(X_i - \theta)^2}{K}\right)^{-2} - \left(1 + \frac{(X_i - \theta)^2}{K}\right)^{-1} \right]$$

```
% Initialize
theta_init = 0;
maxiter = 50;
tol = 1e-9;

% Vector to store estimates
theta_mle = NaN(S, 1);

% Newton-Raphson iterations
for s = 1:S
   old_theta = theta_init;
   new_theta = theta_init;
   dist = 1;
   iter = 1;

   % Get the sample for this simulation
   x_sam = X(:, s);

   while (iter < maxiter && dist > tol)
       % Get new iteration
       % Definitions of gradient and hessian are at the end of the code
       new_theta = old_theta - hessian_t(x_sam, K, old_theta) \ grad_t(x_sam, K, old_theta);

       % Update distance
       dist = (new_theta - old_theta)^2;

       iter = iter + 1;
       old_theta = new_theta;
   end

   theta_mle(s) = new_theta;
end

% Plot the distribution of theta_mle
clf;
histogram(theta_mle);
hold on;
ylims = ylim;
plot([theta theta], [ylims(1) ylims(2)]);
title('Sampling Distribution of $\hat{\theta}$', 'interpreter', 'latex');
```




```
function g = grad_t(x, K, theta)
    g = 0;

    for i = 1:length(x)
        g = g + (x(i) - theta) / (1 + (x(i) - theta)^2 / K);
    end

    g = g * (K + 1) / K;
end

function H = hessian_t(x, K, theta)
    H = 0;

    for i = 1:length(x)
        H = H + ...
            2 * (x(i) - theta)^2 / K * (1 + (x(i) - theta)^2 / K)^(-2) - ...
            (1 + (x(i) - theta)^2 / K);
    end

    H = H * (K + 1) / K;
end
```