Quantitative Risk and Asset Management, Lesson 2

Maximum likelihood; Optimization

Paolo Giordani

BI, Department of Finance

Spring 2020

Ordinary Least Squares review

- Given the linear model $y_i = \beta^T x_i + \epsilon_i$, OLS estimates β as the solution to

$$\min_{\widehat{\beta}} \sum_{i=1}^{N} (y_i - \widehat{\beta}^T x_i)^2 \equiv \sum e_i^2,$$

with solution

$$\mathbf{b} = \sum_{i=1}^{N} (x_i x_i^T)^{-1} \sum_{i=1}^{N} x_i y_i,$$

or, in matrix notation,

$$\mathbf{b} = (X^T X)^{-1} X^T \mathbf{y}.$$

- The variance of ϵ_i is estimated as $\widehat{\sigma}^2 = \frac{1}{N} \sum e_i^2$, but OLS makes no explicit assumption about the distribution, and to compute VaR, ES etc... we need a full distribution. It is common to assume that $\epsilon_i \sim N(0, \widehat{\sigma}^2)$. However, normality is a very poor assumption for most financial returns. To move further and estimate other distributions, we need more general tools, starting with Maximum Likelihood.

Nonlinear least squares and machine learning

- Most statistical learning/machine learning models/algorithms for a continuous variable y_i can be framed as nonlinear regression problems

$$y_i = f_\beta(x_i) + \epsilon_i,$$

where $f_\beta(x_i)$ is a function of x_i and of a paramter vector β. Regression splines, regression trees, and neural networks are all examples of different ways of building a function $f_\beta(x_i)$.

- The most common way of training/estimating these models is by nonlinear least squares (where feasible)

$$\min_{\widehat{\beta}} \sum_{i=1}^{N} \left(y_i - f_\beta(x_i)\right)^2 \equiv \sum e_i^2,$$

which has no analytical solution in general but must be optimized numerically. Notice how all the interest is in modeling $E(y|x) = f_\beta(x_i)$, with no specific assumption being made about the distribution of ϵ_i. In forecasting, if a confidence interval is required, it is then again assumed that $\epsilon_i \sim N(0, \widehat{\sigma}^2)$.

- Most of machine learning and statistical learning is typically interested in models for $E(y|x)$.
- The maximum likelihood framework we are about to study does cover this case. However, our main interest in maximum likelihood is to develop better models of $p(y_i|x_i)$ and in particular of $p(\epsilon_i|x_i)$. That's become most financial variables are not close to Gaussian, and therefore risk measures like VaR, ES etc.. built on the Gaussian are built poorly (and typically underestimate worst-case losses).

Maximum likelihood

- A general approach to estimate any model defined by a probability density function.
- Nearly automatic: Specify: a) the log pdf as a function of parameters – $\log p(y|\theta)$ – and b) a starting value for θ (for the maximization algorithm).
- Produces estimates of the parameter vector θ that are: (a) unbiased or nearly, (b) maximally efficient asymptotically.
- (a) means that, in finite sample, we can expect $E(\theta_{ML}) \approx \theta_{TRUE}$, where the expectation is over (hypothetical) repeated samples. The approximation becomes increasingly good as the sample size n gets larger.
- (b) means that as the sample gets sufficiently large, $E(\theta_{ML} - \theta_{TRUE})^2 \leq E(\widehat{\theta} - \theta_{TRUE})^2$ for any other estimator $\widehat{\theta}$ (IF the model is correctly specified!).

Probability density function

- We start with a model that can be defined by a probability density function $p(\mathbf{y}|\mathbf{x}, \theta)$, where $\mathbf{y} = \{y_1, ..., y_n\}$. We'll omit dependence on $\mathbf{x}$ from now on and write $p(\mathbf{y}|\theta)$.
- Example: the Gaussian linear model

$$y_i = \beta^T x_t + \epsilon_i, \ \epsilon_i \sim N(0, \sigma^2)$$

has $\theta = \{\beta, \sigma^2\}$ and

$$y_t \sim N(\beta^T x_t, \sigma^2),$$

so

$$p(y_t|\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y_t - \beta^T x_t)^2}$$

Probability density function: remarks

- Unlike OLS, the assumption that $\epsilon_i \sim N(0, \sigma^2)$ is now part of the model.
- Example: If we instead specify a t-distribution with v degrees of freedom, so $\epsilon_t \sim t(0, \sigma^2, v)$, we have $\theta = \{\beta, \sigma^2, v\}$ and

$$y_t \sim t(\beta^T x_t, \sigma^2, v),$$

  which will give different estimates of β and σ^2 compared to the Gaussian assumption.
- The IID assumption is also part of the model.
- The asymptotic properties of Maximum Likelihood (asymptotically unbiased and efficient estimates) rely on a correctly specified model.

Factorizing the density

- To go from $p(y_t|\theta)$ to $p(\mathbf{y}|\theta)$, it is typically convenient and computationally efficient to *factorize the density* as

$$p(\mathbf{y}|\theta) = \prod_{i=1}^{n} p(y_t|\theta).$$

- In factorizing, we have assumed that the model is correctly specified so that $y_t|x_t$ is independent of $y_j|x_j$ for $t > j$.
- We'll always work with the log density, in which case the factorization is

$$\log p(\mathbf{y}|\theta) = \sum_{i=1}^{n} \log p(y_t|\theta).$$

From log density to log likelihood

- The likelihood $L(\theta; \mathbf{y})$ has the same mathematical form as the density $p(\mathbf{y}|\theta)$, but the likelihood interprets the observations $\mathbf{y}$ as fixed (the function is from θ to $p(\mathbf{y}|\theta)$).
- Example: we can write

$$p(y_t|\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y_t - \beta' x_t)^2},$$

and

$$L(\theta; y_t) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y_t - \beta' x_t)^2}.$$

Maximum likelihood (ML)

- Maximum Likelihood asks (kind of): what is the likelihood of any given value θ conditional on having observed the data $\mathbf{y}$?
- Maximum likelihood estimates θ as the value that maximizes this log-likelihood, that is

$$\theta_{ML} = \arg\max_{\theta} l(\theta; \mathbf{y}),$$

  where $l(\theta; \mathbf{y}) = \log L(\theta; \mathbf{y})$.
- Typically requires solving a numerical optimization problem. Closed-form solutions only in special cases.

Linear model with Gaussian errors

- The Gaussian linear model (notice the change in notation from β^T to β'):

$$y_i = \beta' x_i + \epsilon_i, \ \epsilon_i \sim N(0, \sigma^2)$$

has $\theta = \{\beta, \sigma^2\}$ and log-likelihood (omitting terms that do not depend on parameters)

$$\begin{aligned} I(\theta; y) &= -\frac{1}{2} \sum_{i=1}^{n} \left(\log \sigma^2 + \frac{1}{\sigma^2} (y_i - \beta' x_i)^2 \right) = \\ &= -\frac{n}{2} \log \sigma^2 - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \beta' x_i)^2. \end{aligned}$$

- A bit of algebra proves that $\widehat{\beta}_{ML} = \widehat{\beta}_{OLS} = \left(\sum_{i=1}^{n} x_i x_i' \right)^{-1} \sum_{i=1}^{n} x_i y_i$, and $\widehat{\sigma}^2_{ML} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \widehat{\beta}'_{ML} x_i)^2 = \widehat{\sigma}^2_{OLS}$.
- Conclusion: OLS is asympotically optimal in a well-defined sense for Gaussian errors ("squares" and "normality" have a strong bond).

Asymptotic distribution of ML estimators

- One of the most celebrated results in statistics states that, as $n \longrightarrow \infty$

$$\theta_{ML} \overset{a}{\sim} N(\theta_0, \mathbf{I}(\theta_0)^{-1}),$$

where $\theta_0 = \theta_{TRUE}$, and $\mathbf{I}(\theta_0)$ can be estimated consistently (that is, with increasingly small errors as the sample size grows) by the *observed information matrix* (which is minus the Hessian of the log-likelihood)

$$\mathbf{I}(\theta_0) \longrightarrow \widehat{\mathbf{I}}(\theta_{ML}) = -\sum_{i=1}^{n} \frac{\partial^2 l(\theta_{ML}; y_i)}{\partial \theta_{ML} \partial \theta'_{ML}}$$

Hypothesis testing and "plugging-in"

- The standard use of this result in frequentist statistics and econometrics is only for hypothesis testing. Asymptotically, $\left((\widehat{\theta}_j - \theta_j)/se(\widehat{\theta}_j)\right) \sim N(0,1)$, where $se(\widehat{\theta}_j) = \sqrt{\left[\widehat{\mathbf{I}}(\widehat{\theta})^{-1}\right]_{jj}}$, θ_j is the hypothetical true parameter value, and where "asymptotically" means roughly "approximately, where the approximation improves with the sample size". For example, if $|\widehat{\theta}_j/se(\widehat{\theta}_j)| > 1.96$, the parameter $\widehat{\theta}_j$ is said to be "statistically different from zero at a confidence level of 5%".
- IMPORTANT: In frequentist statistics, the parameter vector is considered fixed (not a random quantity with a distribution), while any estimate $\widehat{\theta}_j$ is a random quantity with a distribution. When it comes to using the model, say for forecasting or computing measures of risk like VaR and ES, the ML estimate $\widehat{\theta}$ is then typically *plugged into* the model, and $p(y|\widehat{\theta})$ is then used. This is very convenient but, from a Bayesian perspective (which we'll cover in future classes), neglects the uncertainty around our estimate of θ.

- When discussing Bayesian methods, we'll see that they account for parameter uncertainty, and do not plug-in a single estimate.
- For now, remember the point that typically, in statistics, econometrics, machine learning etc... the distribution of θ_{ML} is not used when it comes to making any forecast, whether a point forecast (conditional mean $E(y|x)$) or an interval forecast (VaR).

Maximum likelihood in practice (software)

- In R, you can performs general minimization by calling the function *optim( )*. Or you can do maximum likelihood with the function *mle( )*, which calls *optim( )*. In both cases, I suggest you start with *method = "BFGS"*. More on this soon (when discussing optimization).
- Of course, many models are already coded in R (Python, Matlab...) and you can just call the appropriate function, but we want to have at least the ability of writing our own model.
- As a minimum, all you need to specify is: a) a function to compute the log likelihood (or minus the log-likelihood), b) a starting value for the parameter vector θ.
- For large models it is helpful (in some cases even necessary) to code analytical derivatives of the *gradient (gradient=*$\frac{\partial l(\theta;y)}{\partial \theta}$*)*. All good optimizers and mle functions/packages give you the option of coding analytical derivatives. If you don't, numerical derivatives are used.

Maximum likelihood in practice (software)

- Try to avoid loops: define the log-likelihood using matrix expressions for maximum speed (unless you code e.g. in C++, Fortran or Julia).
- Transforming parameters is sometimes helpful. For example, I typically maximize wrt $\log \sigma^2$ rather than σ^2. Good transformation make the likelihood less asymmetric (see lab session). Notice that $\max l(\theta; y)$ and $\max l(g(\theta); y)$ have the same solution if $g(\cdot)$ is a one-to-one function.
- Coding the likelihood L and then taking its log may lead to underflow ($\log(L)$ inaccurate if L is too small). Bad practice. Code the log-likelihood directly, or code the $(n, 1)$ vector of $l(\theta; y_i)$, which, again, is the same as $p(y_i|x_i, \theta)$, then take logs of this vector, and them sum the logs up.
- If $Var(\widehat{\theta})$ is needed, compute the numerical Hessian at $\widehat{\theta}_{ML}$. This involves no extra coding (see lab).
- The *mle( )* functions in R and Matlab have this option built-in. Be careful with signs if coding minus the log-likelihood.

Introduction to optimization

- Maximum likelihood is, in principle, extremely general and easy to code: just write a function to compute the log-likelihood and a function to compute starting values, and you are set to go.
- That's great as we can spend our time thinking of good models.
- The solution is, however, not available analytically in general. The optimization must be performed numerically. R code will do this for you, but you need to have some understanding of what is going on in the background because there is no guarantee that the optimizer will find the global maximum of the likelihood function.

- Optimization is a big, big topic. Here we'll just sketch an introduction tailored specifically to maximum likelihood. What is specific about maximum likeliood optimization (as opposed to general optimization) is that we are maximizing a sum of a large number of terms, since $\log p(\mathbf{y}|\theta) = \sum_{i=1}^{n} \log p(y_t|\theta)$, and therefore we can often expect the log-likelihood to be approximately quadratic in θ for large enough n (due to a central limit theorem kicking in), that is, approximately Gaussian.[1]
- This introduction is meant to make you more informed users of optimization algorithms.

[1] See the slide "Asymptotic Distribution of the ML Estimator". A Gaussian distribution for θ_{ML} implies a quadratic form for the log-likelihood and viceversa. More on this when looking at Newton-Raphson optimization.

General advice on likelihood optimization

1. Put some efforts on starting values. Avoid "silly" starting values. The closer to the solution we start, the better behaved the likelihood will be in general.
2. Many optimization algorithms may not work as well if parameters have widely different size (orders of magnitude different). It can help to standardize covariates (e.g. use *scale* in R) and take logs of parameters where needed (e.g. $\log(\sigma^2)$ instead of σ^2).
3. Constraints on parameters complicate matters *a lot*. If at all possible, try to transform parameters so that an unconstrained optimization can be performed. e.g. $\log(\sigma^2)$ instead of σ^2 with $\sigma^2 > 0$.

A very simple classification of optimization algorithms

	uses derivatives	**derivative-free**
deterministic	steepest descent; Newton-Raphon;	Nelder-Mead
	quasi-NR (BFGS);	
stochastic	gradient based stochastic approximation:	random search;
	(mostly stochastic steepest descent)	simulated annealing

Several of these options are available in *optim* and *mle* in R (See e.g. https://stat.ethz.ch/R-manual/R-devel/library/stats/html/optim.html): Nelder-Mead, BFGS, and SAN. *optim* and *mle* uses Nelder-Mead as a default, but method = "BFGS" provides a much better default for most maximum likelihood problems.
Derivative-free optimization is in general only feasible when the number of parameters is small. In the maximum-likelihood context, I would only recommend it if the number of parameters is small but you suspect the log-likelihood is very poorly behaved (very far from Gaussian), or as a starting value for derivative-based methods.

Steepest descent (or "gradient descent") (deterministic)

- Note: most optimizers are set-up as *minimizers* (hence the most common name is *steepest descent*, not *steepest ascent*) for a loss function, which in our case is minus the log-likelihood. Make sure to know whether the function you are using is minimizing (code minus log-lik.) or maximizing (code log-lik.). I will assume we are maximizing the log-likelihood.
- Steepest ascent initializes the vector θ at some starting value θ_0 and then moves it in the direction of the gradient (the steepest direction)

$$\begin{aligned} \theta_{k+1} &= \theta_k + \alpha g_k, \text{ where} \\ g_k &= \frac{\partial l(\cdot)}{\partial \theta_k}, \end{aligned}$$

and α is a scalar (can be a vector in more sophisticated implementations).

- There is no obvious one-way of setting α. Common choices are: i) set it to a fairly small number, e.g. 0.05, ii) do a line search and, at each step, set α to maximize the objective function (i.e. keeps moving in the direction of the gradient as long as the log-likelihood keeps improving).
- The iteration terminates when $||g_k - g_{k-1}|| < \epsilon$ for a small ϵ, e.g. $1e-07$, or some similar criterion.
- This is the traditional approach in neural network estimation/training, where it is called *backpropagation.*

Analytical and numerical derivatives

- Computation costs for an analytical gradient is typically similar to one log-likelihood evaluations.
- If the analytical gradient is not provided, numerical derivatives are used. The i-th element of the gradient is approximated by computing

$$[g_k]_i = \frac{\partial l(\cdot)}{\partial [\theta_k]_i} \approx \frac{l(\theta_k + \delta \mathbf{i}) - l(\theta_k - \delta \mathbf{i})}{2\delta},$$

  where δ_i is a vector with 1 in the i-th row.
- This requires 2 evaluations of the log-likelihood. Hence computing the numerical gradient costs $2p$ evaluations of the log-likelihood, where $p = \dim(\theta)$ is the number of parameters. This is often not a big issue if p is small, but increases computation costs dramatically if p is large.
- For some models, there are packages that will compute analytical derivatives automatically (e.g. Sympy in Python).

Stochastic gradient descent: A one-slide introduction

- Stochastic gradient ascent is a special case of stochastic optimization. It iterates on

$$\theta_{k+1} = \theta_k + \alpha_k \widetilde{g}_k$$

where, in the likelihood context, $\widetilde{g}_k$ is the gradient computed on a random (small) sub-set of the data (different at every k), and α_k is a scalar or vector with certan properties, like $\alpha_k > 0$, $\alpha_k \longrightarrow 0$ for $k \longrightarrow \infty$ and $\sum_{k=1}^{\infty} \alpha_k^2 < \infty$.

- This can lead to faster convergence compared to deterministic gradient descent (although determining convergence is more difficult), particularly when n is very large.
- The idea has been around for several decades, but substantial improvements have been made in recent years (e.g. *Adam* algorithm). This is the main approach in the estimation/training of large neural networks (deep learning). We will not use it.

Newton-Raphson

The NR algorithm is based on a quadratic approximation of the log-likelihood $l(\theta)$ (short for $l(\theta; y)$) around a value θ_k (where k indexes the iteration, so the algorithm is initialized at θ_0). Consider a *second-order Taylor expansion* of $l(\theta)$ around θ_k

$$\begin{aligned} l(\theta) &\approx l(\theta_k) + \frac{\partial l(\cdot)}{\partial \theta_k}(\theta - \theta_k) + \frac{1}{2}(\theta - \theta_k)^T \frac{\partial^2 l(\cdot)}{\partial \theta_k \partial \theta_k^T}(\theta - \theta_k) = \\ &= l(\theta_k) + g_k(\theta - \theta_k) + \frac{1}{2}(\theta - \theta_k)^T H_k(\theta - \theta_k) \end{aligned}$$

We want to find the value of θ which **maximizes** this expression. The FOC wrt θ is

$$g_k + H_k(\theta - \theta_k) = 0,$$

with solution

$$\theta^* = \theta_k - H_k^{-1} g_k. \tag{1}$$

We set $\theta_{k+1} = \theta^*$ and continue to iterate on (1) until $\theta_{k+1} \simeq \theta_k$. If the condition is achieved, the algorithm has converged to a *local* mode.

Comments on Newton-Raphson

- If $l(\theta)$ is quadratic in θ, there is one mode and convergence is in one step from any starting value...
- ... and maximum likelihood estimates have an asymptotic Gaussian distribution (the log Gaussian is quadratic in θ).
- In finite samples, $l(\theta)$ will not be exactly quadratic, but there is is a powerful reason to expect NR type algorithm to perform well for likelihood maximization, especially for "well-behaved" problems (model fits the data well, large sample, good starting values ...).
- The closer θ_k is to the maximimum, the better the approximation (because it is more local) for all smooth functions. Once we are fairly close to the maximum, NR is very efficient (most efficient actually).

Drawbacks of Newton-Raphson

- The Hessian must be computed and inverted at every step. If the Hessian is computed numerically, roughly p^2 evaluations of the log-likelikhood are required, where p is the dimension of the vector θ (number of parameters). This is slow for large p. If the Hessian is available analytically, we still need to invert it at every iteration. If p is very large, this becomes a problem.
- Far from the mode, $l(\theta)$ may not be close to quadratic, and NR can take large steps in the wrong direction and generally be unstable and unreliable. Reducing the step by setting $\lambda = 0.5$ in $\theta_{k+1} = \theta_k - \lambda H_k^{-1} g_k$ or evaluating $l(\theta)$ with a *line search* (or *grid search*) (different values of λ) can help, unless H_k^{-1} is not defined or extremely large (e.g. a plateau).
- In summary, NR has maximum efficiency in the best-case scenario (which we expect in large samples), but is fragile in less rosy scenarios.

- *Steepest descent* sets $H_k^{-1} \equiv I$. The optimization is now less fragile and may work even with thousands of parameters (as in neural networks), but convergence is very slow in general. An intermediate form is to pretend that the Hessian H_k is diagonal (and compute only the p rows of its diagonal matrix).
- Quasi Newton-Raphson methods attempt to make the optimization more reliable while preserving much of the efficiency.

BFGS (most common quasi-NR algo)

- BFGS performs the NR iterations, but replaces H_k^{-1} by an approximation Q_k which is updated from iteration to iteration (via the Matrix Inversion Lemma) using only gradient evaluations. Q_0 (corresponding to starting value θ_0) can be initialized as the identity matrix I (in which case BFGS starts out as steepest descent).
- Q_k is always positive definite, so that each step of the iteration is well defined.
- Only gradient evaluations are performed, which requires $2p$ evaluations of the likelihood instead of p^2. This makes each iteration faster when numerical derivatives are used. Also, no matrix inversion is required.
- A line search is typycall performed at each step (but not necessarily).
- Quite reliable in converging to a local mode. If the likelihood is well behaved (as it should in large samples), there is only one mode.
- If the number of parameters is in the hundreds or thousands, this will probably not work, and steepest descent (deterministic or stochastic) is the natural option.

Keywords (for repetition)

OLS (and why not enough)
ML asymptotically unbiased and max efficient
Gaussian density (formula)
factorizing the density/likelihood
ML asymptotic distribution
gradient; Hessian
"plug-in" estimates
general advice on likelihood maximization

optimization algorithms
Newton-Raphson
Steepest descent
Line search
BFGS
Stochastic optimization

Assignments (will be covered in lab session by Jonas)

Write code to a) simulate a dataset of n observations from the DGP (data-generating process)

$$\begin{aligned} x_t &\sim N(0,1), \\ y_t &= \beta_0 + \beta_1 x_t + \epsilon_t \\ \epsilon_t &\sim N(0,\sigma^2), \end{aligned}$$

b) code the log-likelihood corresponding to the density $p(y_t|x_t)$ and estimate the parameter vector $\theta = \{\beta_0, \beta_1, \sigma^2\}$ by maximum likelihood (so we can compare with the analytical solution, which is least squares), c) compare $\widehat{\beta}_0, \widehat{\beta}_1, \widehat{\sigma}$ with the OLS estimates and issue a warning if there is any non-trivial difference (a sign of failed maximization), d) plot the log-likelihood evaluated at $\widehat{\beta}_0, \widehat{\beta}_1$ and varying σ^2 in the region $[\widehat{\sigma}^2 - 2se(\widehat{\sigma}), \widehat{\sigma}^2 + 2se(\widehat{\sigma})]$.

1) Set $\beta_0 = 1, \beta_1 = 0.5, \sigma^2 = 1$. Play around with the sample size n, the starting values and with the maximization algorithms (BFGS and SAN are enough).

Assignments (continued).

2) In step d), you should notice that the plot has a smooth quadratic shape for large enough n , but for sufficiently small n the log-likelihood drops down much faster on the left (so the Gaussian asymptotic approximation is not accurate).

3) Now repeat the entire process but use the log of σ^2, so $\theta = \{\beta_0, \beta_1, \log(\sigma^2)\}$. For a given sample, the log of the ML estimate of $\widehat{\sigma}^2$ should be the same as the ML estimate of $\log(\widehat{\sigma}^2)$. However, the plot in d) should be near-quadratic even for very low n (that is, the Gaussian asymptotic approximation works better after this transformation).