Optimisation

Birgit Erni

2021-03-18

Contents

These notes have evolved from original notes by Professor Theo Stewart. Sections on solving with Excel have disappeared, some algorithms have been added.

1 Optimization Methods in Statistics

Numerical optimization techniques have been used in statistical research for a long time, but have become much more accessible in recent years due to the availability of easy-to-use optimization software, such as the *Solver* tool in Excel, and numerous optimization functions in R and other software.

Suppose that we have postulated a model of the form $y = \phi(\mathbf{x}; \boldsymbol{\theta})$, where $\mathbf{x}$ is the predictor (often a vector), and $\boldsymbol{\theta}$ the unknown parameter (or vector of parameters). Typically, we would observe a random sample $(y_i, \mathbf{x}_i), i = 1, \ldots, n$, on the basis of which we would seek to obtain an "estimate'' of $\boldsymbol{\theta}$. The standard frequentist approach is to set up the corresponding likelihood function, which is maximized. It is the exception rather than the rule for it to be possible to find the maximum analytically by differentiating and setting to zero. We thus in most cases need to find the maximum by numerical search means.

For example, consider the following hypothetical model:

$$y = x^{\alpha} + \beta x + \text{ error}$$

If the errors are approximately homoscedastic and normal, the maximum likelihood boils down to minimization of:

$$\sum_{i=1}^{n} (y_i - x_i^{\alpha} - \beta x_i)^2$$

Differentiation with respect to α and β is still possible, but solving for α and β to make the derivative zero is not so easy.

%In Excel, we start by setting up a sheet containing the data, and two cells to contain guessed values for α and β. We can include calculations of the predicted values corresponding to each x_i, the deviations, their squares, and the sum of squares. Such a worksheet is illustrated in Figure~**??**, which also contains some sample data (generated by simulation from the above model with $\alpha = 0.5$ and $\beta = 1$, with observations at $x = 30$ and at $x = 75$ contaminated by additional error).

As the least squares criterion can be sensitive to certain outlying observations, there is sometimes an advantage in seeking to minimize the sum of absolute deviations:

$$\sum_{i=1}^{n} |y_i - x_i^{\alpha} - \beta x_i|$$

also called the L_1 norm. In using Solver for this problem, a numerically more stable approach (avoiding problems arising from non-differentiability at 0) is obtained by defining $2n$ deviational variables, δ_i^+ and δ_i^- for $i = 1, \ldots, n$, and introducing n constraints of the form:

$$y_i - x_i^{\alpha} - \beta x_i = \delta_i^+ - \delta_i^-.$$

This rather roundabout approach is particularly useful if the model can be expressed in a form which is linear in the parameters, as then the numerically extremely robust and stable linear programming algorithm can be invoked. It is sometimes useful to exploit this numerical robustness, even when there is say one non-linearizable parameter, by running the model for a sequence of fixed values for this parameter. For example, in the above illustration we ran the linear optimization option for a sequence of values for α. The results are summarized in the following table:

α	Solver results	
	Optimal β	Sum of Absolute deviations
0.40	1.041	22.37
0.50	0.998	20.29
0.55	0.972	19.59
0.60	0.939	19.09
0.65	0.898	18.72
0.70	0.845	18.59
0.75	0.779	18.60
0.80	0.697	18.81
0.90	0.467	21.64

Clearly the optimal solution (best estimate) for α is around 0.70–0.75, with an associated value for β interpolated from the above table. Running the non-linear solver on the full problem, i.e. optimizing over α and β, yielded the optimal solution $\alpha = 0.723$, $\beta = 0.816$, giving a sum of absolute deviations of 18.57.

The use of the formulation in terms of deviational variables and of the linear programming solver can be exploited in another way. It is also possible to find the model fit which minimizes the *maximum deviation* at all observation points. The maximum deviation can be defined by:

$$\Delta = \max_{i=1}^{n}(\delta_i^+ + \delta_i^-).$$

Minimization of Δ is easily achieved using the linear programming option, simply by including constraints:

$$\Delta \geq \delta_i^+ + \delta_i^-$$

for each $i = 1, 2, \ldots, n$, and setting minimization of Δ as the target.

Optimization is closely linked to solving nonlinear equations, or finding roots. For example, to find the MLE we find the score equation $\ell(\theta)$ and solve $\ell(\theta) = 0$. In other words the maximum of g is a solution to $g'(x) = 0$.

To solve a set of linear equations with constraints linear programming techniques are used, not covered here.

Each optimization problem is unique, and therefore it is important to understand enough about the methods to be able to tweak the

algorithms. We will look at some methods for univariate problems and then some for multivariate problems. All of these are *iterative numerical* procedures which converge towards a solution.

2 *The Bisection method*

Consider the following problem of maximising

$$g(x) = \frac{logx}{1+x}$$

with respect to x. We can find the first derivative, but solving for $g'(x) = 0$ analytically is not possible.

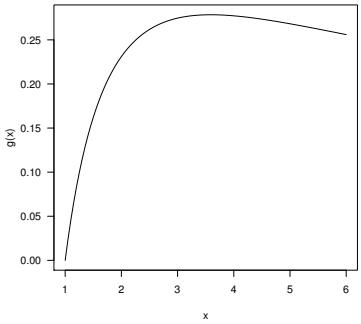


Figure 1: The maximum of $g(x) = \frac{logx}{1+x}$ occurs at $x^* \approx 3.59112$.

Starting Value: Choose points a_0 and b_0 so that they bracket the maximum. $x^{(0)} = (a_0 + b_0)/2$

Updating Equations: at iteration $t + 1$

$$\begin{aligned}(a_{t+1}, b_{t+1}) =& (a_t, x^{(t)}) \qquad \text{if} \qquad g'(a_t)g'(x^{(t)}) \leq 0 \\ & (x^{(t)}, b_t) \qquad \text{if} \qquad g'(a_t)g'(x^{(t)}) > 0\end{aligned}$$

and

$$x^{(t+1)} = \frac{a_{t+1} + b_{t+1}}{2}$$

until convergence. One advantage of the bisection method is that it does not require the second derivative g''.

3 *Convergence Criteria and Stopping Rules*

Numerical optimization procedures are iterative and can approach the solution very slowly. Therefore they need to be stopped when the solution is close enough (has converged). The convergence criteria

are checked with every iteration, and once met the last $x^{(t+1)}$ is taken to be the solution.

When should we stop the algorithm?

There are several possibilities:

1. $g'(x) \approx 0$. Not ideal, because we can still have large changes in x if $g(x)$ is very flat.
2. Small changes in x. Not ideal, because small changes in x are relative (depends on scale of x).
3. $g'(x) \approx 0$. This is used more as a check, rather than as a stopping criterion.
4. $g'(x) \approx 0$
5. $g(\theta^{(k)}) - g(\theta^{(k-1)})$
6. $|\theta^{(k)} - \theta^{(k-1)})| < \epsilon$

Here, ϵ is a constant, tolerable imprecision, often referred to as **tolerance**.

7. Relative convergence criterion:

$$\frac{|\theta^{(k)} - \theta^{(k-1)}|}{|\theta^{(k)}|} < \epsilon$$

We typically assess convergence by monitoring $|x^{(t+1)} - x^{(t)}|$ and use $g'(x^{(t+1)})$ as a backup check.

The relative convergence criterion

$$\frac{|\theta^{(k)} - \theta^{(k-1)}|}{|\theta^{(k)}|} < \epsilon$$

allows us to specify a target precision (e.g. within 1%) without worrying about the units of x.

4 *Newton's Method*

Newton's method is based on a Taylor series expansions around a:

$$g(x^*) = g(a) + (x^* - a)g'(a) + (x^* - a)^2 \frac{g''(a)}{2!} \dots$$

For example, if we want to maximise $g(x)$ or equivalently want to solve $g'(x) = 0$, we can approximate $g'(x)$ by a linear function around $x^{(t)}$, where $x^{(t)}$ is the current estimate.

$$g'(x^*) \approx g'(x) + (x^* - x)g''(x)$$

This approximation is then used to find the new solution $x^{(t+1)}$, which is that x^* for which $g'(x^*) = 0$.

$$x^{(t+1)} = x^{(t)} - \frac{g'(x^{(t)})}{g''(x^{(t)})}$$

Exercise: Approximate $g(x)$ by a quadratic (2nd order) Taylor series expansion around $x^{(t)}$, maximise this analytically, and show that this leads to exactly the same updating equation.

For Maximum Likelihood Estimation Newton's method becomes

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\ell'(\theta^{(t)})}{\ell''(\theta^{(t)})}$$

Newton's method can also be used for multivariate likelihood functions. A special case of this is the **Fisher Scoring** algorithm, where we approximate $-l''(\theta)$ by $I(\theta)$:

$$\theta^{(t+1)} = \theta^{(t)} + \ell'(\theta^{(t)})I^{-1}(\theta^{(t)})$$

$I(\theta)$ is the expected Fisher information, evaluated at θ.

5 *Ascent Algorithms for multivariate problems*

Take the updating equation from Newton's algorithm and let $M^{(t)}$ be the matrix of second derivatives $g''(x_{ij}^{(t)})$, the Hessian matrix. Then

$$x^{(t+1)} = x^{(t)} - [M^{(t)}]^{-1} g'(x^{(t)})$$

The gradient indicates the steepest direction uphill. The second derivative gives an indication to which side the maximum (minimum) lies, and the faster the rate of change (of the slope) the smaller the step should be.

If calculation of M is too expensive or difficult, substitute $-I$, the negative identity matrix and the updating equations become

$$x^{(t+1)} = x^{(t)} + g'(x^{(t)})$$

6 *Nonlinear Gauss-Seidel Iteration*

This method can be used to optimize a system of k nonlinear equations in k unknowns. In each iteration each of the k components is updated one at a time by solving:

$$g_j'(x_j^{(t+1)}) = 0$$

7 *Optimization in R: optim()*

1. `optim(par, fn, gr, method, control, hessian)`

- fn: function to be minimized
- par: initial parameter guess
- gr: gradient function; only needed for some methods
- method: defaults to "Nelder-Mead" (which does not use gradients)

- control: optional list of control settings (maximum number of iterations, tolerance for convergence)
- hessian: should the final Hessian be returned? default FALSE

For both 'nlm' and 'optim' the function to be optimized must be defined with its first argument the *vector of parameters*.

2. `nlm(f, p, ..., hessian = FALSE)`

3. `optimize(f, interval, ..., lower = min(interval), upper = max(interval), maximum = FALSE)`

   Use `optimize()` for one-dimensional (single parameter) problems.

4. `uniroot()` finds the roots of a function (where equals zero). This can be used to solve for maxima / minima, by solving for the value of x where the derivative function equals zero (numerically).

8 References

1. Givens, G. & Hoeting, J. (2005). Computational statistics. Wiley

9 Optimisation Prac

9.1 Question 1: Bisection Method

1. Use the bisection method to find the maximum of $g(x)$:

$$g(x) = \frac{log(x)}{1+x}$$

2. Check your answer: $x \approx 3.5912$
3. Check your answer using the `uniroot()`.

9.2 *Question 2: Maximum likelihood*

Assume that these 13 observations are from a Poisson distribution, with rate parameter λ:

```
counts <- c(3, 1, 1, 3, 1, 4, 3, 2, 0, 5, 0, 4, 2)
```

1. Set up the likelihood and write a function to calculate the likelihood as a function of the parameter λ.
2. Plot the likelihood function.
3. Have a look at the examples at the bottom of the help pages for `optim()` and `nlm()`.
4. Use `optim()` and `nlm()` (non-linear minimization) to find the MLE for λ.
5. Use Newton's method (from scratch) to find the MLE for λ in the above problem. You should get exactly the same answer as with the `optim` and `nlm` functions. You should also be able to check from your plot whether your answer makes sense.
6. Add the MLE to a plot of your function.
7. Add a quadratic approximation of the likelihood function, around an initial estimate, to your plot.
8. Can you find an estimate of the standard error for your λ estimate?
9. λ = average rate. $\hat{\lambda} = \bar{y}$.

9.3 *Guidelines for translating this into code*

How does one go about programming this, e.g. the Newton algorithm, but hopefully these guidelines will work for other methods.

- Start simple and in small steps. Don't start at the beginning but at the most crucial step, then build around that.
- What do you need? The algorithm centers around the updating equation, how to update your current estimate to a new, better estimate.

- Check what is required for this equation. The updating equation requires: a function for the likelihood (or negative likelihood if you minimize), i.e. it should return the likelihood given a value of λ. You will also need functions that return the first and second derivatives of the likelihood given a value of lambda.
- This is what your functions should look like: your first argument should be the parameter (or vector of parameters), and then add arguments to explicitly define all the values requires to calculate the likelihood and its derivatives. `y` is the vector of observations. Give your functions good names that you can recognize.

```
nll <- function(lambda, y) {

}

d1 <- function(lambda, y) {

}

d2 <-
```

- Use log-likelihoods instead of likelihoods! Why?
- Plot the likelihood: create a sequence of lambda values. For each calculate the corresponding likelihood, and plot. Your sequence should have very fine resolution so that your function looks like a continuous function. Don't be stingy when it comes to good plots!
- Now test your updating step by choosing an initial value for lambda, based on the plot of your likelihood function.
- What next? Do you need a for loop or a while loop? When do you need to stop the algorithm? Translate this into code.
- One of the more difficult parts of such an algorithm is that you need to keep track of your current estimate and your previous estimate (for the convergence criterion). Check carefully that these get updated at the correct points. Give these variables names that are easy to understand so that you don't get confused by which is which.
- Newton algorithm in a nutshell:
  - initial value
  - while not converged

- update

More advice:

- be systematic
- think: what do I need for this step, which variables need to be defined first
- think in small steps, one thing at a time
- rather add an extra step than try and do multiple things at once
- add comments for YOU to remember crucial steps
- If I get stuck I do a lot of printing, if need be for every step I print out the value. You can also check this in the environment window. This helps a lot with checking if R is doing what it should. And then run line-by-line
- Before you run the whole loop set to values to something specific, and run one iteration of the loop.

9.4 Question 3: Constrained (linear) optimization

Maximize $2x + 2y + 3z$ subject to

$$\begin{aligned} -2x + y + z &\leq 1 \\ 4x - y + 3z &\leq 3 \\ x \geq 0, y \geq 0, z &\geq 0 \end{aligned}$$

`constrOptim(theta, f, grad, ui, ci)`: Look at its help function and the example at the bottom of the help page.

For `constrOptim` the constraints need to be rephrased such that

$ui \cdot \theta - ci >= 0$, where θ is the parameter vector.

9.5 Question 4: Maximum Likelihood using `nlm` and Gauss-Seidel

For the following data assume the model:

$$Y_i \sim Poisson(\lambda_i) \qquad \lambda_i = \alpha + \beta x_i$$

```
y <- c(2,4,3,0,1,4,3,6,10,7)

x <- c(0.49909145, 1.24373850, 0.34376255, 0.03833630, 0.09699331,
       0.19469526, 0.21237902, 1.56276200, 1.56909233, 1.88487024)
```

1. Use R's `nlm()` function to maximise the likelihood, give parameter estimates and their standard errors. Note that here there is a parameter vector: two parameters.
2. Use the Gauss-Seidel algorithm to maximise the likelihood.
3. Do you get the same answer?
4. What about Newton's method for this problem? (Hint: ℓ' is a vector, ℓ'' is a matrix.)
5. When would you use Gauss-Seidel, when Newton's method?
6. You can check your answer by fitting the model using R's `glm()` function (note, no log link).

10 Appendix

10.1 Taylor series expansions

Taylor series expansions can be used to approximate functions. For example, a second order Taylor series expansion gives a quadratic approximation of a function $g(x)$. In the following example we find a quadratic approximation of $g(x)$ based on values calculated at $x = a$. The approximation is linked to the true function at $x = a$, but approximated for all other values of x.

$$g(x) = g(a) + (x - a)g'(a) + (x - a)^2 \frac{g''(a)}{2!} \ldots$$

For maximising functions, e.g. to maximise $g(x)$ we can use a Taylor series expansion and then solve $g'(x) = 0$. The advantage of this is that quadratic functions are a lot easier to maximise than

some other functions. In the above, we maximise the approximation, iteratively, rather than the function directly.

We can also use a Taylor series expansion to approximate $g'(x)$ by a linear function around $x^{(t)}$, our current estimate.

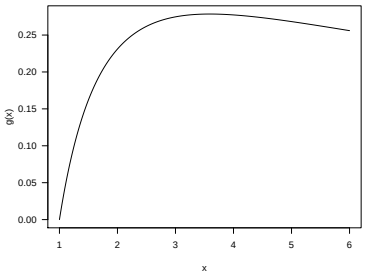
$$g'(x) \approx g'(x) + (x^* - x)g''(x)$$

To find a new solution, we solve for x^* where $g'(x^*) = 0$:

This leads to exactly the same algorithm:

$$x^{(t+1)} = x^{(t)} - \frac{g'(x^{(t)})}{g''(x^{(t)})}$$

Exercise: Approximate $g(x)$ by a quadratic (2nd order) Taylor series expansion around $x^{(t)}$ and show that this leads to the same algorithm when finding the maximum of $g(x)$.



10.2 Reminders

- maximize $f(x)$ = minimize $-f(x)$
- at an *interior* minimum/maximum: $f'(x) = 0$
- at an *interior* minimum $f''(x) > 0$
- if h is strictly increasing, e.g. log, then min $f(x)$ = min $h(f(x))$
- local vs global minimum/maximum
- in $>=$ 2 dimensions, matrix of second derivatives = **Hessian** matrix (positive definite at minimum = all eigenvalues > 0)

10.3 Tips for Optimization

Good *starting values* are important in many problems. It is important to ensure that the estimate is a global and not a local maximum / minimum. For this it is important to plot the function!

Always plot the function first (where possible).

10.4 Hessian matrix

The Hessian matrix is the matrix of second derivatives. In likelihood theory, the observed information matrix $I = -H$. This can be used to obtain standard errors (variance estimates) of parameters.

10.5 Computer Rounding Error

```
(log(0.01^200))
(200 * log(0.01))
```

Computers round very small decimal values to zero. You should do statistical calculations using sum of logs, rather than taking product of a large number of probabilities.