Solving Nonlinear Equations & Optimization (One Dimension)

**Problem:** for a function $f(x)$, find $x_0$ such that $f(x_0) = 0$. 

![Graph showing a function $f(x)$ with a point where $f(x_0) = 0$.]
One Root: The Bisection Method

This one’s guaranteed to converge (at least to a singularity, if not an actual root).

1. Start with $a$ and $b$ such that $f(a)$ and $f(b)$ are opposite signs.

2. Choose midpoint $c = a + (b - a)/2$.

3. If $f(c)$ has a sign opposite of $a$, then set $b = c$. Otherwise, set $a = c$.

4. Repeat until desired tolerance is attained.
One Root: Brent’s Method

- Brackets with a local quadratic interpolation of three points.

- At a given iteration, if the next computed point falls outside of the bracketing interval, a bisection step is used.

- Is the method underlying `uniroot()` in R.

- More details in Press et al (1992). (Brent’s is the most is the method most highly recommended by NR for single nonlinear root-finding.)
One Root: Newton’s Method

Local linear approximation using $f'(x)$. Steps:

- With first guess $x_0$, compute $f'(x_0)$ (slope of approximating line).
- Next guess $x_1$ is the root of the tangent line extending from $f(x_0)$.
- Iterate until convergence.
## A Comparison

<table>
<thead>
<tr>
<th>Method</th>
<th>Requires $f'(x)$?</th>
<th>Guaranteed?</th>
<th>Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bisection</td>
<td>No</td>
<td>Yes</td>
<td>Linear</td>
</tr>
<tr>
<td>Brent’s</td>
<td>No</td>
<td>Almost</td>
<td>Superlinear</td>
</tr>
<tr>
<td>Newton’s</td>
<td>Yes</td>
<td>No</td>
<td>Quadratic*</td>
</tr>
</tbody>
</table>

* If “close.”

These same relative trade-offs exist for higher-dimensional procedures.
Optimization in One Dimension

**Problem**: for a function $f(x)$, find $x_m$ such that $f(x_m) > f(x)$ (or $f(x_m) < f(x)$) for all $x \neq x_m$. We’ll focus on minima, since finding a max for $f(x)$ is equivalent to finding a min for $-f(x)$.

Global versus local:

- Multiple extrema.
- Boundaries.
One-dimensional: Golden Section Search

- An analogue to the bisection method for finding roots.
- Proceeds as follows:
  - Begin with 3 points $x_1 < x_2 < x_3$, that are thought to contain a local minimum.
  - Choose new point $x_0$, such that $x_1 < x_0 < x_3$.
  - Form a new bracketing interval based on the relative values of $f(x_0)$ and $f(x_2)$. For example, if $x_0 < x_2$, then the new interval is $(x_0, x_3)$ if $f(x_0) > f(x_2)$, or it’s $(x_1, x_2)$ if $f(x_0) < f(x_2)$.
  - Iterate until convergence.
What does “Golden” mean?

The question is: following the steps on the previous slide, how do we select $x_0$?

The answer is: we make a choice that guarantees a proportional reduction in the width of the interval at each step. For example, if $x_0 < x_2$, then for this to happen regardless of the value of $f(x_0)$ we need to satisfy $x_0 - x_1 = x_3 - x_2 = \alpha(x_3 - x_1)$, where $\alpha$ represents the proportion of the interval eliminated at each step.

To get the same reduction at the next iteration, the points also must satisfy $x_2 - x_0 = \alpha(x_3 - x_1) = \alpha[\alpha(x_3 - x_1) + (x_2 - x_0)]$, so $x_2 - x_0 = (x_3 - x_1)\alpha/(1 - \alpha)$. Since $(x_0 - x_1) + (x_2 - x_0) + (x_3 - x_2) = x_3 - x_1$, it follows that $2\alpha + \alpha^2/(1 - \alpha) = 1$, a quadratic whose only solution satisfying $0 < \alpha < 1$ is

$$\alpha = (3 - \sqrt{5})/2.$$  

Hence, the proportion of the interval remaining after each iteration is given by

$$1 - \alpha = (\sqrt{5} - 1)/2 \approx 0.618,$$

which is known as the Golden Mean.
How do we use the value $\alpha$?

- Start with an interval $[x_1, x_3]$ thought to contain the min.

- Select the interior points $x_0 = x_1 + \alpha(x_3 - x_1)$ and $x_2 = x_3 - \alpha(x_3 - x_1)$.

- Evaluate $f(x_0)$ and $f(x_2)$.
  - If $f(x_0) < f(x_2)$, new interval is $[x_1, x_2]$ and next point selected is $x_1 + \alpha(x_2 - x_1)$.
  - If $f(x_0) > f(x_2)$, new interval is $[x_0, x_3]$ and next point selected is $x_3 - \alpha(x_3 - x_0)$.

- Iterate.
Brent’s Method

- Works in a manner analogous to Brent’s for root-finding: local quadratic interpolation, with a safety net in case new points fall outside of the bracket.

- Too complicated to describe here (a lot of housekeeping computations), although you can find out more in NR.

- The method used by R’s `optimize()` function.
Solving Several Nonlinear Equations

The problem is to find solutions for a system of the form

\[ f_1(x_1, x_2, \ldots, x_p) = 0 \]
\[ f_2(x_1, x_2, \ldots, x_p) = 0 \]
\[ \vdots \]
\[ f_p(x_1, x_2, \ldots, x_p) = 0 \]
Options

- Multivariate Newton’s, or Newton-Raphson (NR).
- Modified NR – line searches and backtracking.
- Multivariate secant method – Broyden’s method.

Similar trade-offs apply (as we discussed with one equation) in terms of convergence and knowledge of the Jacobian.
Why is finding several roots such a problem?

• “There are *no* good, general methods for solving systems of more than one nonlinear equation” – from *NR*.

• Often, functions \( f_1, f_2, \ldots, f_p \) have nothing to do with each other.

• Finding solutions means identifying where the \( p \) zero contours in the \( p-1 \) zero hypersurfaces simultaneously intersect.

• These can be difficult to home in on without some insight into how the \( p \) functions relate to one another.

• See example on following slide, with \( p = 2 \).
Reproduced from *Numerical Recipes*: 
Developing a multivariate linear approximation:

Let $f$ denote the entire vector of $p$ functions, and let $x=(x_1,\ldots,x_p)$ denote an entire vector of values $x_i$, for $i=1,\ldots,p$.

Taylor series expansion of $f_i$ in a neighborhood of $x$:

$$f_i(x + \delta x) = f_i(x) + \sum_{j=1}^{p} \frac{\partial f_i}{\partial x_j} \delta x_j + O(\delta x^2).$$

Note that the partial derivatives in this equation arise from the Jacobian matrix $J$ of $f$. So in matrix notation we have:

$$f(x + \delta x) = f(x) + J \cdot \delta x + O(\delta x^2).$$
Newton-Raphson

From expansion on previous slide, neglecting terms of order $\delta x^2$ and higher, and setting equal to zero we obtain a set of linear equations for the corrections $\delta x^2$ that move each function simultaneously closer to zero:

$$\mathbf{J} \cdot \delta \mathbf{x} = -\mathbf{f},$$

which can be solved using $LU$ decomposition. This gives us an iterative approach correcting and updating a solution:

$$\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} + \delta \mathbf{x},$$

which we can iterate to convergence (i.e., how close either the 1-norm or $\infty$-norm of $\delta \mathbf{x}$ is to zero).
Evaluating the Jacobian

• As we often cannot easily evaluate the Jacobian analytically, a conventional option is numerical differentiation.

• Numerical evaluation of the Jacobian relies on finite difference equations. Approximate value of the \((i,j)\)th element of \(J(x)\) is given by:

\[
J_{ij} \approx \frac{f_i(x + h_j e_j) - f_i(x)}{h_j},
\]

where \(h_j\) is some very small number and \(e_j\) represents a vector with 1 at the \(j\)th position and zeroes everywhere else.
Modified Newton-Raphson

Note that a full Newton step can be represented as

$$\delta x = -J^{-1} \cdot f.$$ 

When we are not close enough to the solution, this is not guaranteed to decrease to decrease the value of the function. How do we know if we should take the full step?

One strategy is to require that the step decrease the inner product of $f$, which is the same requirement as trying to minimize $f = (f \cdot f)/2$.

Another is to note that the Newton step is a descent direction:

$$\nabla f \cdot \delta x = (f \cdot J) \cdot (-J^{-1} \cdot f) = -f \cdot f < 0.$$
Modified Newton-Raphson (continued)

Strategy:

i. Define $p = \delta x$, and a Newton iteration as

$$x_{\text{new}} = x_{\text{old}} + \lambda p,$$

where a full Newton step specifies $\lambda = 1$.

ii. If $f$ is reduced, then go to next iteration.

iii. If $f$ is not reduced, then backtrack, selecting some $\lambda < 1$.

Value of $\lambda$ for a conventional backtrack is selected to ensure that the average rate of decrease is at least some fraction of the initial rate of decrease, and that rate of decrease of $f$ at new value of $x$ is some fraction of the rate for the old value of $x$. 
Multidimensional Optimization

The problem: find a minimum for the function

\[ f(x_1, \ldots, x_p). \]

Note that in many statistical applications the functions we wish to optimize (e.g., loglikelihoods) are convex, and hence fairly well behaved.

Also, in terms of the various approaches, options involve trade-offs between rate of convergence and information about the gradient and Hessian. (The latter two can often be numerically evaluated.)
Strategies

1. Newton-Raphson (applied to the gradient).


3. Powell’s Method.


5. Variable Metric Methods.
Nelder-Mead Simplex Approach

- *Simplex* is a figure with \( p+1 \) vertices in \( p \) dimensions (a triangle in two dimensions, or a tetrahedron in three dimensions).

- Start with a set of \( p+1 \) points that define a *finite* simplex (i.e., one having finite volume).

- Simplex method then takes a series of reflective steps, moving the “highest” point (where the \( f \) is largest) through the opposite face of the simplex to a lower point.

- Steps are designed to preserve the volume, but simplex may expand (lengthen) where feasible to facilitate convergence.

- When simplex reaches a “valley floor”, it takes contractive steps.

- *NR* implementation descriptively refers to this routine as “amoeba”.
Possible simplex moves:
Powell’s Method (aka, Direction Set Methods)

- We know how to minimize a single nonlinear equation.
- Given a one-dimensional approach, a direction set method proceeds as follows:
  - Start at a point \( x_0 = (x_1, \ldots, x_p) \).
  - Consider a set of vector directions \( n_1, n_2, \ldots, n_p \) (e.g., these might arise from the gradient of \( f \)).
  - In the direction \( n_1 \), find the scalar that minimizes \( f(x_0 + \lambda n_1) \) (using a one-dimensional method).
  - Replace \( x_0 \) with \( x_0 + \lambda n_1 \).
  - Iterate through \( n_2, \ldots, n_p \), and continue iterating until convergence.

- Note that you can use whatever nonlinear optimization routine you want – say, Brent’s or the Golden Section Search.
Conjugate Gradient Methods

• If you can compute the gradient, it turns out that you can enjoy substantial computational savings over a direction set method.

• Idea is to choose directions based on the gradient, but it turns out that the path of steepest descent (i.e., given a current guess $x_i$ for the minimum, the path of steepest descent is the negative gradient evaluated at $x_i$) is not a good direction. (See figure on slide following.)

• Instead, a set of conjugate directions are derived such that the we will not just proceed down the new gradient, but in a direction that is conjugate to the old gradient – and conjugate to all previous directions traversed.

• Note: given the symmetric hessian $H$, two vectors $x_i$ and $n_j$ are said to be conjugate if $x_i’Hn_j = 0$. 
Problems with Steepest Descent

(a) In a long, narrow “valley”, steepest descent takes many steps to reach the valley floor. (b) For a single (magnified) step, direction begins perpendicular to contours, but winds up parallel to local contours where minimum is reached.
Quasi-Newton Methods

- Similar to conjugate gradient methods, in the sense that we are accumulating information from \( p \) successive line minimizations using gradient information to find the minimum of a quadratic form.

- Quasi-Newton methods can be thought of as a means of applying Newton-Raphson to the gradient, without the need for the Hessian.

- Using N-R with the gradient, given a current guess \( x_i \) the next guess is given by:

\[
x_{i+1} = x_i - H^{-1}\nabla f(x_i).
\]

- Note that with quasi-Newton, we start out with a positive-definite matrix used as an approximation to the Hessian. Successive iterations update this approximation, which converges to the actual Hessian.

- Most common implementations of this approach are so-called Davidon-Fletcher-Powell (DFP) and Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithms.
Newton-Raphson in R:

Non-Linear Minimization

Description

This function carries out a minimization of the function \( f \) using a Newton-type algorithm. See the references for details.

Usage

\[
\text{nlm}(f, p, \text{hessian = } \text{FALSE}, \text{typsize = } \text{rep(1, length(p))}, \text{fscale = 1, print.level = 0, ndigit = 12, gradtol = 1e-6, stepmax = } \text{max(1000 * } \sqrt{\text{sum((p/typsize)^2))}, 1000)}, \text{steptol = 1e-6, iterlim = 100, check.analyicals = TRUE, } \ldots)
\]

Arguments

- \( f \)
  - the function to be minimized. If the function value has an attribute called \( \text{gradient} \) or both \( \text{gradient} \) and \( \text{hessian} \) attributes, these will be used in the calculation of updated parameter values. Otherwise, numerical derivatives are used. \text{deriv} returns a function with suitable \( \text{gradient} \) attribute. This should be a function of a vector of the length of \( p \) followed by any other arguments specified by the \( \ldots \) argument.

- \( p \)
  - starting parameter values for the minimization.

- \text{hessian}
  - if \text{TRUE}, the \text{hessian} of \( f \) at the minimum is returned.

- \text{typsize}
  - an estimate of the size of each parameter at the minimum.

- \text{fscale}
  - an estimate of the size of \( f \) at the minimum.

- \text{print.level}
  - this argument determines the level of printing which is done during the minimization process. The default value of 0 means that no printing occurs, a value of 1 means that initial and final details are printed and a value of 2 means that full tracing information is printed.

- \text{ndigit}
  - the number of significant digits in the function \( f \).

- \text{gradtol}
  - a positive scalar giving the tolerance at which the scaled gradient is considered close enough to zero to terminate the algorithm. The scaled gradient is a measure of the relative change in \( f \) in each direction \text{p[1]} divided by the relative change in \text{p[1]}.

- \text{stepmax}
  - a positive scalar which gives the maximum allowable scaled step length. \text{stepmax} is used to prevent steps which would cause the optimization function to overflow, to prevent the algorithm from leaving the area of interest in parameter space, or to detect divergence in the algorithm. \text{stepmax} would be chosen small enough to prevent the first two of these occurrences, but should be larger than any anticipated reasonable step.

- \text{steptol}
  - a positive scalar providing the minimum allowable relative step length.

- \text{iterlim}
  - a positive integer specifying the maximum number of iterations to be performed before the program is terminated.

- \text{check.analyicals}
  - a logical scalar specifying whether the analytic gradients and Hessians, if they are supplied, should be checked against numerical derivatives at the initial parameter values. This can help detect incorrectly formulated gradients or Hessians.

- \ldots
  - additional arguments to \( f \).
Simplex and Quasi-Newton Methods in R:

**General-purpose Optimization**

**Description**

General-purpose optimization based on Nelder–Mead, quasi-Newton and conjugate-gradient algorithms. It includes an option for box-constrained optimization and simulated annealing.

**Usage**

```r
optim(par, fn, gr = NULL,
      method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN"),
      lower = -Inf, upper = Inf,
      control = list(), hessian = FALSE, ...)
```

**Arguments**

- **par**
  Initial values for the parameters to be optimized over.

- **fn**
  A function to be minimized (or maximized), with first argument the vector of parameters over which minimization is to take place. It should return a scalar result.

- **gr**
  A function to return the gradient for the "BFGS", "CG" and "L-BFGS-B" methods. If it is NULL, a finite-difference approximation will be used.
  For the "SANN" method it specifies a function to generate a new candidate point. If it is NULL a default Gaussian Markov kernel is used.

- **method**
  The method to be used. See **Details**.

- **lower, upper**
  Bounds on the variables for the "L-BFGS-B" method.

- **control**
  A list of control parameters. See **Details**.

- **hessian**
  Logical. Should a numerically differentiated Hessian matrix be returned?

- **...**
  Further arguments to be passed to `fn` and `gr`. Beware of partial matching to earlier arguments.