Unit 11: Optimization

November 5, 2018

References:

- Gentle: *Computational Statistics*
- Lange: *Optimization*
- Monahan: *Numerical Methods of Statistics*
- Givens and Hoeting: *Computational Statistics*
- Materials online from Stanford's EE364a course on convex optimization, including Boyd and Vandenberghe's (online) book Convex Optimization, which is also linked to from the course webpage.

1 Notation

We'll make use of the first derivative (the gradient) and second derivative (the Hessian) of functions. We'll generally denote univariate and multivariate functions (without distinguishing between them) as $f(x)$ with $x = (x_1, \ldots, x_p)$. The (column) vector of first partial derivatives (the gradient) is $f'(x) = \nabla f(x) = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_p})^\top$ and the matrix of second partial derivatives (the Hessian) is

$$f''(x) = \nabla^2 f(x) = H_f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_p} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_p} \\ \vdots & \vdots & \ddots & \\ \frac{\partial^2 f}{\partial x_1 \partial x_p} & \frac{\partial^2 f}{\partial x_2 \partial x_p} & \cdots & \frac{\partial^2 f}{\partial x_p^2} \end{pmatrix}.$$

In considering iterative algorithms, I'll use $x_0,\ x_1, \ldots, x_t,\ x_{t+1}$ to indicate the sequence of values as we search for the optimum, denoted x^*. x_0 is the starting point, which we must choose (often carefully). If it's unclear at any point whether I mean a value of x in the sequence or a sub-element of the x vector, let me know, but hopefully it will be clear from context most of the time.

I'll try to use x (or if we're talking explicitly about a likelihood, θ) to indicate the argument with respect to which we're optimizing and Y to indicate data involved in a likelihood. I'll try to use z to indicate covariates/regressors so there's no confusion with x.

2 Overview

The basic goal here is to optimize a function numerically when we cannot find the maximum (or minimum) analytically. Some examples:

1. Finding the MLE for a GLM

2. Finding least squares estimates for a nonlinear regression model,

$$Y_i \sim \mathcal{N}(g(z_i;\beta), \sigma^2)$$

   where $g(\cdot)$ is nonlinear and we seek to find the value of $\theta = (\beta, \sigma^2)$ that best fits the data.

3. Maximizing a likelihood under constraints

4. Fitting a machine learning prediction method

Maximum likelihood estimation and variants thereof is a standard situation in which optimization comes up.

We'll focus on **minimization**, since any maximization of f can be treated as minimization of $-f$. The basic setup is to find the *argument*, x, that minimizes $f(x)$:

$$\arg\min_{x\in D} f(x)$$

where D is the domain. Sometimes $D = \Re^p$ but other times it imposes constraints on x. When there are no constraints, this is unconstrained optimization, where any x for which $f(x)$ is defined is a possible solution. We'll assume that f is continuous as there's little that can be done systematically if we're dealing with a discontinuous function.

In one dimension, minimization is the same as root-finding with the derivative function, since the minimum of a differentiable function can only occur at a point at which the derivative is zero. So with differentiable functions we'll seek to find x s.t. $f'(x) = \nabla f(x) = 0$. To ensure a minimum, we want that for all y in a neighborhood of x^*, $f(y) \leq f(x^*)$, or (for twice differentiable functions) $f''(x^*) = \nabla^2 f(x^*) = H_f(x^*) \geq 0$, i.e., that the Hessian is positive semi-definite.

Different strategies are used depending on whether D is discrete and countable, or continuous, dense and uncountable. We'll concentrate on the continuous case but the discrete case can arise in statistics, such as in doing variable selection.

In general we rely on the fact that we can evaluate f. Often we make use of analytic or numerical derivatives of f as well.

To some degree, optimization is a solved problem, with good software implementations, so it raises the question of how much to discuss in this class. The basic motivation for going into some of the basic classes of optimization strategies is that the function being optimized changes with each problem and can be tricky to optimize, and I want you to know something about how to choose a good approach when you find yourself with a problem requiring optimization. Finding global, as opposed to local, minima can also be an issue.

Note that I'm not going to cover MCMC (Markov chain Monte Carlo) methods, which are used for approximating integrals and sampling from posterior distributions in a Bayesian context and in a variety of ways for optimization. If you take a Bayesian course you'll cover this in detail, and if you don't do Bayesian work, you probably won't have much need for MCMC, though it comes up in MCEM (Monte Carlo EM) and simulated annealing, among other places.

Goals for the unit Optimization is a big topic. Here's what I would like you to get out of this:

1. an understanding of line searches (one-dimensional optimization),
2. an understanding of multivariate derivative-based optimization and how line searches are useful within this,
3. an understanding of derivative-free methods,
4. an understanding of the methods used in R's optimization routines, their strengths and weaknesses, and various tricks for doing better optimization in R, and
5. a basic idea of what convex optimization is and when you might want to go learn more about it.

3 Univariate function optimization

We'll start with some strategies for univariate functions. These can be useful later on in dealing with multivariate functions.

3.1 Golden section search

This strategy requires only that the function be unimodal.

Assume we have a single minimum, in $[a, b]$. We choose two points in the interval and evaluate them, $f(x_1)$ and $f(x_2)$. If $f(x_1) < f(x_2)$ then the minimum must be in $[a, x_2]$, and if the converse in $[x_1, b]$. We proceed by choosing a new point in the new, smaller interval and iterate. At each step we reduce the length of the interval in which the minimum must lie. The primary question involves what is an efficient rule to use to choose the new point at each iteration.

Suppose we start with x_1 and x_2 s.t. they divide $[a, b]$ into three equal segments. Then we use $f(x_1)$ and $f(x_2)$ to rule out either the leftmost or rightmost segment based on whether $f(x_1) < f(x_2)$. If we have divided equally, we cannot place the next point very efficiently because either x_1 or x_2 equally divides the remaining space, so we are forced to divide the remaining space into relative lengths of 0.25, 0.25, and 0.5. The next time around, we may only rule out the shorter segment, which leads to inefficiency.

The efficient strategy is to maintain the *golden ratio* between the distances between the points using $\phi = (\sqrt{5} - 1)/2 \approx .618$, the golden ratio. We start with $x_1 = a + (1 - \phi)(b - a)$ and $x_2 = a + \phi(b - a)$. Then suppose $f(x_1) < f(x_2)$. We now choose to place x_3 s.t. it uses the golden ratio in the interval $[a, x_1]$: $x_3 = a + (1 - \phi)(x_2 - a)$. Because of the way we've set it up, we once again have the third subinterval, $[x_1, x_2]$, of equal length as the first subinterval, $[a, x_3]$. The careful choice allows us to narrow the search interval by an equal proportion,$1 - \phi$, in each iteration. Eventually we have narrowed the minimum to between x_{t-1} and x_t, where the difference $|x_t - x_{t-1}|$ is sufficiently small (within some tolerance - see Section 4 for details), and we report $(x_t + x_{t-1})/2$. We'll see an example of this on the board in class.

3.2 Bisection method

The bisection method requires the existence of the first derivative but has the advantage over the golden section search of halving the interval at each step. We again assume unimodality.

We start with an initial interval (a_0, b_0) and proceed to shrink the interval. Let's choose a_0 and b_0, and set x_0 to be the mean of these endpoints. Now we update according to the following algorithm, assuming our current interval is $[a_t, b_t]$:

$$[a_{t+1}, b_{t+1}] = \begin{cases} [a_t, x_t] & \text{if} f'(a_t)f'(x_t) < 0 \\ [x_t, b_t] & \text{if} f'(a_t)f'(x_t) > 0 \end{cases}$$

and set x_{t+1} to the mean of a_{t+1} and b_{t+1}. The basic idea is that if the derivative at both a_t and x_t is negative, then the minimum must be between x_t and b_t, based on the intermediate value theorem.

If the derivatives at a_t and x_t are of different signs, then the minimum must be between a_t and x_t.

Since the bisection method reduces the size of the search space by one-half at each iteration, one can work out that each decimal place of precision requires 3-4 iterations. Obviously bisection is more efficient than the golden section search because we reduce by $0.5 > 0.382 = 1 - \phi$, so we've gained information by using the derivative. It requires an evaluation of the derivative however, while golden section just requires an evaluation of the original function.

Bisection is an example of a *bracketing* method, in which we trap the minimum within a nested sequence of intervals of decreasing length. These tend to be slow, but if the first derivative is continuous, they are robust and don't require that a second derivative exist.

3.3 Newton-Raphson (Newton's method)

3.3.1 Overview

We'll talk about Newton-Raphson (N-R) as an optimization method rather than a root-finding method, but they're just different perspectives on the same algorithm.

For N-R, we need two continuous derivatives that we can evaluate. The benefit is speed, relative to bracketing methods. We again assume the function is unimodal. The minimum must occur at x^* s.t. $f'(x^*) = 0$, provided the second derivative is non-negative at x^*. So we aim to find a zero (a root) of the first derivative function. Assuming that we have an initial value x_0 that is close to x^*, we have the Taylor series approximation

$$f'(x) \approx f'(x_0) + (x - x_0)f''(x_0).$$

Now set $f'(x) = 0$, since that is the condition we desire (the condition that holds when we are at x^*), and solve for x to get

$$x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)},$$

and iterate, giving us updates of the form $x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}$. What are we doing intuitively? Basically we are taking the tangent to $f(x)$ at x_0 and extrapolating along that line to where it crosses the x-axis to find x_1. We then reevaluate $f(x_1)$ and continue to travel along the tangents.

One can prove that if $f'(x)$ is twice continuously differentiable, is convex, and has a root, then N-R converges from any starting point.

Note that we can also interpret the N-R update as finding the analytic minimum of the quadratic Taylor series approximation to $f(x)$.

Newton's method converges very quickly (as we'll discuss in Section 4), but if you start too far from the minimum, you can run into serious problems.

3.3.2 Secant method variation on N-R

Suppose we don't want to calculate the second derivative required in the divisor of N-R. We might replace the analytic derivative with a discrete difference approximation based on the secant line joining $(x_t, f'(x_t))$ and $(x_{t-1}, f'(x_{t-1}))$, giving an approximate second derivative:

$$f''(x_t) \approx \frac{f'(x_t) - f'(x_{t-1})}{x_t - x_{t-1}}.$$

For this variant on N-R, we need two starting points, x_0 and x_1.

An alternative to the secant-based approximation is to use a standard discrete approximation of the derivative such as

$$f''(x_t) \approx \frac{f'(x_t + h) - f'(x_t - h)}{2h}.$$

3.3.3 How can Newton's method go wrong?

Let's think about what can go wrong - namely when we could have $f(x_{t+1}) > f(x_t)$? Basically, if $f'(x_t)$ is relatively flat, we can get that $|x_{t+1} - x^*| > |x_t - x^*|$. We'll see an example on the board and the demo code (see below). Newton's method can also go uphill when the second derivative is negative, with the method searching for a maximum.

First let's see an example of divergence.

```
par(mfrow = c(1,2))
fp <- function(x, theta = 1){
        exp(x*theta)/(1+exp(x*theta)) - .5
}
fpp <- function(x, theta = 1){
        exp(x*theta)/((1+exp(x*theta))^2)
}
xs <- seq(-15, 15, len = 300)

## good starting point
x0 <- 2
xvals <- c(x0,rep(NA,9))
for(t in 2:10){
        xvals[t]=xvals[t-1] - fp(xvals[t-1]) / fpp(xvals[t-1])
}
print(xvals)
```

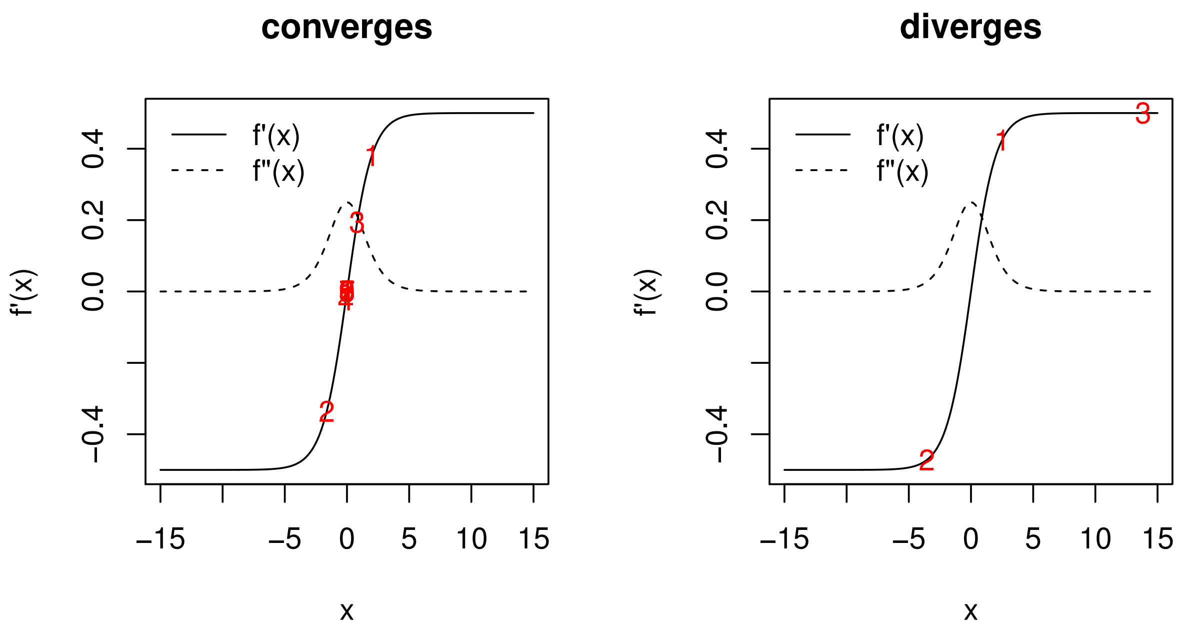
```
##  [1]  2.000000e+00 -1.626860e+00  8.188046e-01 -9.460983e-02  1.412056e-0
##  [6] -4.691188e-13  6.142005e-17  6.142005e-17  6.142005e-17  6.142005e-1
```

```
plot(xs, fp(xs), type = 'l', xlab = 'x', ylab = "f'(x)", main ='converges')
lines(xs, fpp(xs), lty = 2)
legend('topleft', bty = 'n', lty = c(1,2), legend = c("f'(x)", 'f"(x)'))
points(xvals, fp(xvals), pch = as.character(1:length(xvals)), col = 'red')

## bad starting point
x0 <- 2.5
xvals <- c(x0,rep(NA,9))
for(t in 2:10){
        xvals[t]=xvals[t-1] - fp(xvals[t-1]) / fpp(xvals[t-1])
}
print(xvals)
```

```
##  [1]  2.500000e+00 -3.550204e+00  1.384565e+01 -5.152876e+05           In
##  [6]           NaN           NaN           NaN           NaN           Na
```

```
plot(xs, fp(xs), type = 'l', xlab = 'x', ylab = "f'(x)", main = 'diverges')
lines(xs, fpp(xs), lty = 2)
legend('topleft', lty = c(1,2), legend = c("f'(x)", 'f"(x)'), bty = 'n')
points(xvals, fp(xvals), pch = as.character(1:length(xvals)), col = 'red')
```




```
## whoops
```

Now let's see an example of climbing uphill and finding a local maximum rather than minimum.

```
## example of mistakenly climbing uphill

par(mfrow = c(3,1))

# original fxn
f <- function(x) cos(x)
# gradient
fp <- function(x) -sin(x)
# second derivative
fpp <- function(x) -cos(x)
xs <- seq(0, 2*pi, len = 300)

x0 <- 1 # starting point
fp(x0) # negative

## [1] -0.841471

fpp(x0) # negative
```

```
## [1] -0.5403023

x1 <- x0 - fp(x0)/fpp(x0) # whoops, we've gone uphill
## because of the negative second derivative
xvals <- c(x0,rep(NA,9))
for(t in 2:10){
        xvals[t]=xvals[t-1]-fp(xvals[t-1])/fpp(xvals[t-1])
}
xvals

##  [1]  1.000000e+00 -5.574077e-01  6.593645e-02 -9.572192e-05  2.923566e-1
##  [6]  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+0

plot(xs, f(xs), type = 'l', xlab = 'x', ylab = "f'(x)", lwd = 2,
     main = 'uphill to local maximum',)
lines(xs, fp(xs))
lines(xs, fpp(xs), lty = 2)
legend('bottomright', lty = c(1,1,2), lwd = c(2, 1, 1),
       legend = c("f(x)", "f'(x)", 'f"(x)'), bty = 'n')
points(xvals, fp(xvals), pch = as.character(1:length(xvals)), col = 'red')
## and we've found a maximum rather than a minimum...

## in contrast, with better starting points we can find the minimum

x0 <- 2 # ok starting point
fp(x0)

## [1] -0.9092974

fpp(x0)

## [1] 0.4161468

x1 <- x0 - fp(x0)/fpp(x0)
xvals <- c(x0,rep(NA,9))
for(t in 2:10){
        xvals[t]=xvals[t-1]-fp(xvals[t-1])/fpp(xvals[t-1])
}
xvals
```

```
##  [1] 2.000000 4.185040 2.467894 3.266186 3.140944 3.141593 3.141593
##  [8] 3.141593 3.141593 3.141593
```

```
# oscillates and comes close to diverging but converges
plot(xs, f(xs), type = 'l', xlab = 'x', ylab = "f'(x)", lwd = 2,
     main = 'converges to minimum')
lines(xs, fp(xs))
lines(xs, fpp(xs), lty = 2)
legend('bottomright', lty = c(1,1,2), lwd = c(2, 1, 1),
       legend = c("f(x)", "f'(x)", 'f"(x)'), bty = 'n')
points(xvals, fp(xvals), pch = as.character(1:length(xvals)), col = 'red')
```

```
x0 <- 2.5 # good starting point
fp(x0)
```

```
## [1] -0.5984721
```

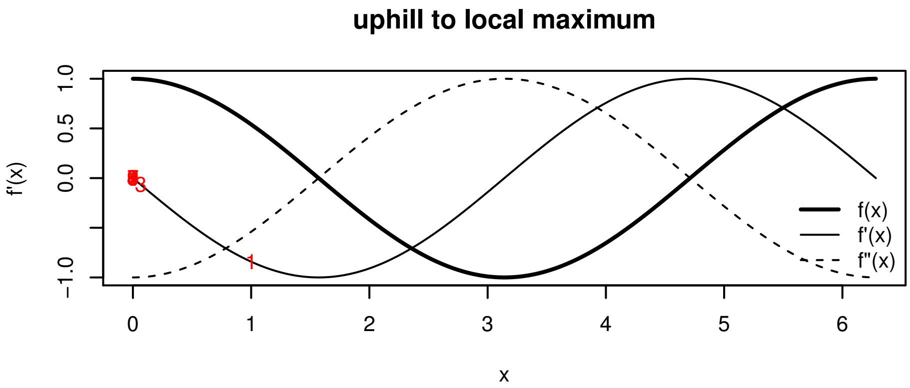
```
fpp(x0)
```

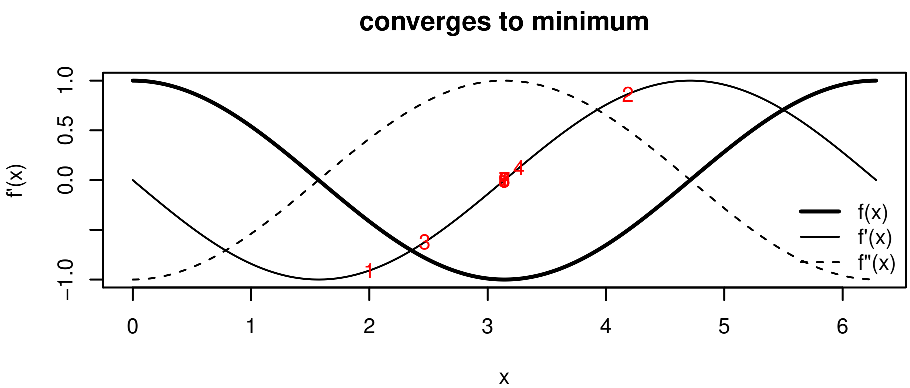
```
## [1] 0.8011436
```

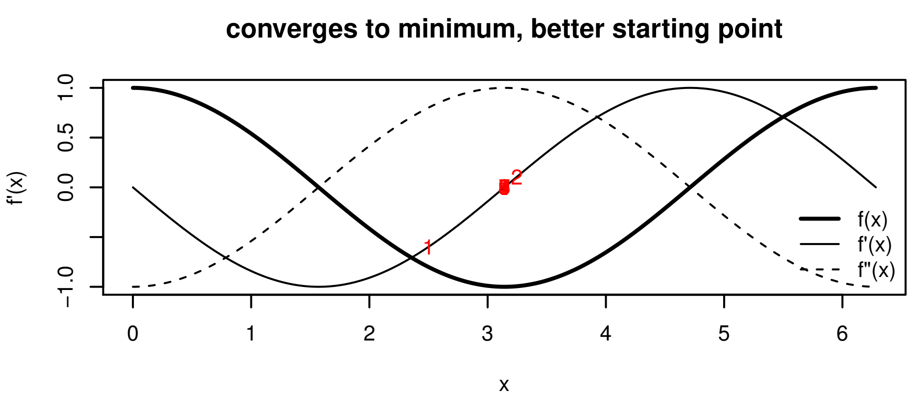
```
x1 <- x0 - fp(x0)/fpp(x0)
xvals <- c(x0,rep(NA,9))
for(t in 2:10){
        xvals[t]=xvals[t-1]-fp(xvals[t-1])/fpp(xvals[t-1])
}
xvals
```

```
##  [1] 2.500000 3.247022 3.141200 3.141593 3.141593 3.141593 3.141593
##  [8] 3.141593 3.141593 3.141593
```

```
## converges quickly
plot(xs, f(xs), type = 'l', xlab = 'x', ylab = "f'(x)", lwd = 2,
     main = 'converges to minimum, better starting point')
lines(xs, fp(xs))
lines(xs, fpp(xs), lty = 2)
legend('bottomright', lty = c(1,1,2), lwd = c(2, 1, 1),
       legend = c("f(x)", "f'(x)", 'f"(x)'), bty = 'n')
points(xvals, fp(xvals), pch = as.character(1:length(xvals)), col = 'red')
```







One nice, general idea is to use a fast method such as Newton's method *safeguarded* by a robust, but slower method. Here's how one can do this for N-R, safeguarding with a bracketing method such as bisection. Basically, we check the N-R proposed move to see if N-R is proposing a step outside of where the root is known to lie based on the previous steps and the gradient values for those steps. If so, we could choose the next step based on bisection.

Another approach is backtracking. If a new value is proposed that yields a larger value of the function, backtrack to find a value that reduces the function. One possibility is a line search but given that we're trying to reduce computation, a full line search is often unwise computationally (also in the multivariate Newton's method, we are in the middle of an iterative algorithm for which we will just be going off in another direction anyway at the next iteration). A basic approach is to keep backtracking in halves. A nice alternative is to fit a polynomial to the known information about that slice of the function, namely $f(x_{t+1})$, $f(x_t)$, $f'(x_t)$ and $f''(x_t)$ and find the minimum of the polynomial approximation.

4 Convergence ideas

4.1 Convergence metrics

We might choose to assess whether $f'(x_t)$ is near zero, which should assure that we have reached the critical point. However, in parts of the domain where $f(x)$ is fairly flat, we may find the derivative is near zero even though we are far from the optimum. Instead, we generally monitor $|x_{t+1}-x_t|$ (for the moment, assume x is scalar). We might consider absolute convergence: $|x_{t+1}-x_t| < \epsilon$ or relative convergence, $\frac{|x_{t+1}-x_t|}{|x_t|} < \epsilon$. Relative convergence is appealing because it accounts for the scale of x, but it can run into problems when x_t is near zero, in which case one can use $\frac{|x_{t+1}-x_t|}{|x_t|+\epsilon} < \epsilon$. We would want to account for machine precision in thinking about setting ϵ. For relative convergence a reasonable choice of ϵ would be to use the square root of machine epsilon or about 1×10^{-8}. This is the *reltol* argument in *optim()* in R.

Problems with the optimization may show up in a convergence measure that fails to decrease or cycles (oscillates). Software generally has a stopping rule that stops the algorithm after a fixed number of iterations; these can generally be changed by the user. When an algorithm stops because of the stopping rule before the convergence criterion is met, we say the algorithm has failed to converge. Sometimes we just need to run it longer, but often it indicates a problem with the function being optimized or with your starting value.

For multivariate optimization, we use a distance metric between x_{t+1} and x_t, such as $\|x_{t+1}-x_t\|_p$, often with $p=1$ or $p=2$.

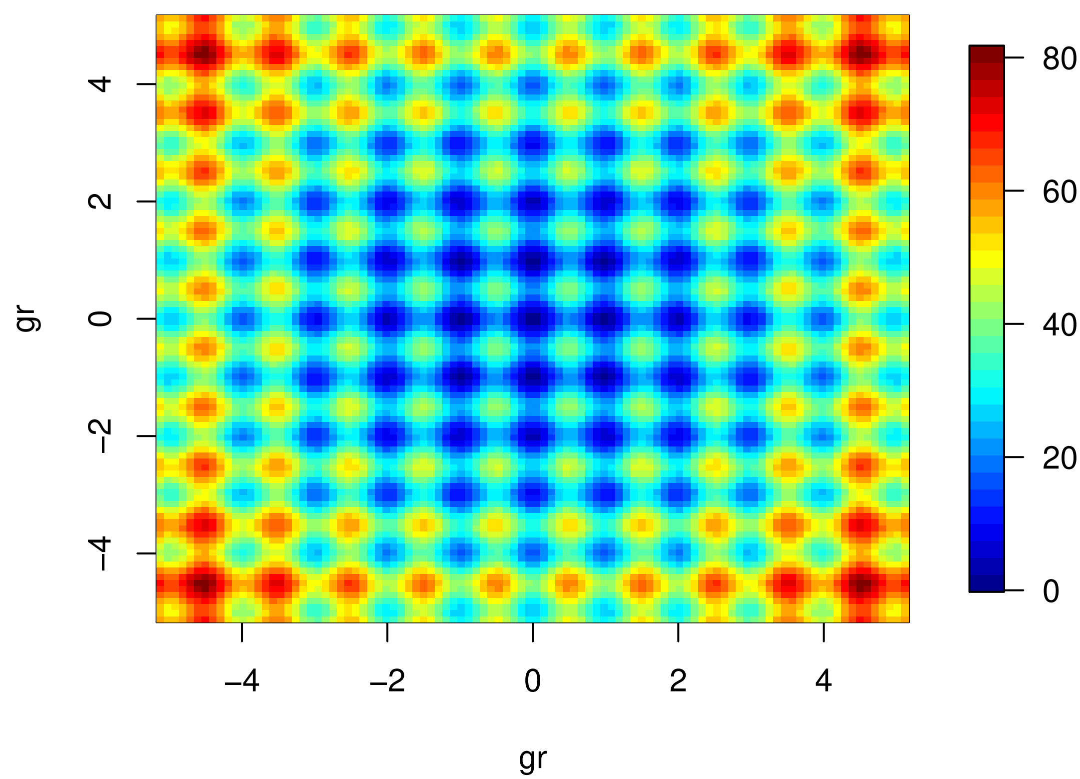
4.2 Starting values

Good starting values are important because they can improve the speed of optimization, prevent divergence or cycling, and prevent finding local optima.

Using random or selected multiple starting values can help with multiple optima (aka multimodality).

Here's a function (the Rastrigin function) with multiple optima that is commonly used for testing methods that claim to work well for multimodal problems. This is a hard function to optimize with respect to, particularly in higher dimensions (one can do it in higher dimensions than 2 by simply making the x vector longer but having the same structure). In particular Rastrigin with 30 dimensions is considered to be very hard.

```
rastrigin <- function(x) {
        A <- 10
        n <- length(x)
        return(A*n + sum(x^2 - A * cos(2*pi*x)))
}
const <- 5.12
nGrid <- 100
gr <- seq(-const, const, len = nGrid)
xs <- expand.grid(x1 = gr, x2 = gr)
y <- apply(xs, 1, rastrigin)
require(fields)
image.plot(gr, gr, matrix(y, nGrid, nGrid), col=tim.colors(32))
```



One R package that may be useful for multi-modal problems is *DEoptim*, which implements an evolutionary algorithm (genetic algorithms are one kind of evolutionary algorithm). It would be interesting to try an evolutionary algorithm on a test function like this.

4.3 Convergence rates

Let $\epsilon_t = |x_t - x^*|$. If the limit

$$\lim_{t\to\infty} \frac{|\epsilon_{t+1}|}{|\epsilon_t|^\beta} = c$$

exists for $\beta > 0$ and $c \neq 0$, then a method is said to have order of convergence β. This basically measures how big the error at the $t+1$th iteration is relative to that at the tth iteration, with the approximation that $|\epsilon_{t+1}| \approx c|\epsilon_t|^\beta$.

Bisection doesn't formally satisfy the criterion needed to make use of this definition, but roughly speaking it has linear convergence ($\beta = 1$), so the magnitude of the error decreases by a factor of c at each step. Next we'll see that N-R has quadratic convergence ($\beta = 2$), which is fast.

To analyze convergence of N-R, consider a Taylor expansion of the gradient at the minimum, x^*, around the current value, x_t:

$$f'(x^*) = f'(x_t) + (x^* - x_t)f''(x_t) + \frac{1}{2}(x^* - x_t)^2 f'''(\xi_t) = 0,$$

for some $\xi_t \in [x^*, x_t]$. Making use of the N-R update equation: $x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}$, and some algebra, we have

$$\frac{|x^* - x_{t+1}|}{(x^* - x_t)^2} = \frac{1}{2}\frac{f'''(\xi_t)}{f''(x_t)}.$$

If the limit of the ratio on the right hand side exists and is equal to c:

$$c = \lim_{x_t \to x^*} \frac{1}{2}\frac{f'''(\xi_t)}{f''(x_t)} = \frac{1}{2}\frac{f'''(x^*)}{f''(x^*)}$$

then we see that $\beta = 2$.

If c were one, then we see that if we have k digits of accuracy at t, we'd have $2k$ digits at $t+1$ (e.g., $|\epsilon_t| = 0.01$ results in $|\epsilon_{t+1}| = 0.0001$), which justifies the characterization of quadratic convergence being fast. In practice c will moderate the rate of convergence. The smaller c the better, so we'd like to have the second derivative be large and the third derivative be small. The expression also indicates we'll have a problem if $f''(x_t) = 0$ at any point [think about what this corresponds to graphically - what is our next step when $f''(x_t) = 0$?]. The characteristics of the derivatives determine the domain of attraction (the region in which we'll converge rather than diverge) of the minimum.

Givens and Hoeting show that using the secant-based approximation to the second derivative in N-R has order of convergence, $\beta \approx 1.62$.

Here's an example of convergence comparing bisection and N-R:

```
options(digits = 10)
f <- function(x) cos(x)
fp <- function(x) -sin(x)
fpp <- function(x) -cos(x)
xstar <- pi # known minimum

## N-R
x0 <- 2
xvals <- c(x0,rep(NA,9))
for(t in 2:10){
        xvals[t] <- xvals[t-1] - fp(xvals[t-1]) / fpp(xvals[t-1])
}
print(xvals)

##  [1] 2.000000000 4.185039863 2.467893675 3.266186278 3.140943912
##  [6] 3.141592654 3.141592654 3.141592654 3.141592654 3.141592654

## bisection
bisecStep <- function(interval, fp){
        xt <- mean(interval)
        if(fp(interval[1]) * fp(xt) <= 0) interval[2] <- xt else interval[1
        return(interval)
}
nIt <- 30
a0 <- 2; b0 <- (3*pi/2) - (xstar - a0)
## have b0 be as far from min as a0 for fair comparison with N-R
interval <- matrix(NA, nr = nIt, nc = 2)
interval[1, ] <- c(a0, b0)
for(t in 2:nIt){
        interval[t, ] <- bisecStep(interval[t-1, ], fp)
}
rowMeans(interval)
```

```
##  [1] 2.785398163 3.178097245 2.981747704 3.079922475 3.129009860
##  [6] 3.153553552 3.141281706 3.147417629 3.144349668 3.142815687
## [11] 3.142048697 3.141665201 3.141473454 3.141569328 3.141617264
## [16] 3.141593296 3.141581312 3.141587304 3.141590300 3.141591798
## [21] 3.141592547 3.141592922 3.141592734 3.141592641 3.141592687
## [26] 3.141592664 3.141592652 3.141592658 3.141592655 3.141592654
```

5 Multivariate optimization

Optimizing as the dimension of the space gets larger becomes increasingly difficult. First we'll discuss the idea of profiling to reduce dimensionality and then we'll talk about various numerical techniques, many of which build off of Newton's method.

5.1 Profiling

A core technique for likelihood optimization is to analytically maximize over any parameters for which this is possible. Suppose we have two sets of parameters, θ_1 and θ_2, and we can analytically maximize w.r.t θ_2. This will give us $\hat{\theta}_2(\theta_1)$, a function of the remaining parameters over which analytic maximization is not possible. Plugging in $\hat{\theta}_2(\theta_1)$ into the objective function (in this case generally the likelihood or log likelihood) gives us the profile (log) likelihood solely in terms of the obstinant parameters. For example, suppose we have the regression likelihood with correlated errors:

$$Y \sim \mathcal{N}(X\beta, \sigma^2\Sigma(\rho)),$$

where $\Sigma(\rho)$ is a correlation matrix that is a function of a parameter, ρ. The maximum w.r.t. β is easily seen to be the GLS estimator $\hat{\beta}(\rho) = (X^\top\Sigma(\rho)^{-1}X)^{-1}X^\top\Sigma(\rho)^{-1}Y$. In general such a maximum is a function of all of the other parameters, but conveniently it's only a function of ρ here. This gives us the initial profile likelihood

$$\frac{1}{(\sigma^2)^{n/2}|\Sigma(\rho)|^{1/2}}\exp\left(-\frac{(Y - X\hat{\beta}(\rho))^{-\top}\Sigma(\rho)^{-1}(Y - X\hat{\beta}(\rho))}{2\sigma^2}\right).$$

We then notice that the likelihood is maximized w.r.t. σ^2 at

$$\hat{\sigma^2}(\rho) = \frac{(Y - X\hat{\beta}(\rho))^\top\Sigma(\rho)^{-1}(Y - X\hat{\beta}(\rho))}{n}.$$

This gives us the final profile likelihood,

$$\frac{1}{|\Sigma(\rho)|^{1/2}}\frac{1}{(\hat{\sigma^2}(\rho))^{n/2}}\exp(-\frac{1}{2}n),$$

a function of ρ only, for which numerical optimization is much simpler.

5.2 Newton-Raphson (Newton's method)

For multivariate x we have the Newton-Raphson update $x_{t+1} = x_t - f''(x_t)^{-1}f'(x_t)$, or in our other notation,

$$x_{t+1} = x_t - H_f(x_t)^{-1}\nabla f(x_t).$$

In class we'll use the demo code in the Unit 11 code file (not shown here) for an example of finding the nonlinear least squares fit to some weight loss data to fit the model (but note that technically speaking one can use profiling in this case, so it's not a perfect example):

$$Y_i = \beta_0 + \beta_1 2^{-t_i/\beta_2} + \epsilon_i.$$

Some of the things we need to worry about with Newton's method in general about are (1) good starting values, (2) positive definiteness of the Hessian, and (3) avoiding errors in deriving the derivatives.

A note on the positive definiteness: since the Hessian may not be positive definite (although it may well be, provided the function is approximately locally quadratic), one can consider modifying the Cholesky decomposition of the Hessian to enforce positive definiteness by adding diagonal elements to H_f as necessary.

5.3 Fisher scoring variant on N-R

The Fisher information (FI) is the expected value of the outer product of the gradient of the log-likelihood with itself

$$I(\theta) = E_f(\nabla f(y)\nabla f(y)^\top),$$

where the expected value is with respect to the data distribution. Under regularity conditions (true for exponential families), the expectation of the Hessian of the log-likelihood is minus the Fisher information, $E_f H_f(y) = -I(\theta)$. We get the observed Fisher information by plugging the data values into either expression instead of taking the expected value.

Thus, standard N-R can be thought of as using the observed Fisher information to find the updates. Instead, if we can compute the expectation, we can use minus the FI in place of the

Hessian. The result is the Fisher scoring (FS) algorithm. Basically instead of using the Hessian for a given set of data, we are using the FI, which we can think of as the average Hessian over repeated samples of data from the data distribution. FS and N-R have the same convergence properties (i.e., quadratic convergence) but in a given problem, one may be computationally or analytically easier. Givens and Hoeting comment that FS works better for rapid improvements at the beginning of iterations and N-R better for refinement at the end.

$$\begin{aligned} (NR):\ \theta_{t+1} &= \theta_t - H_f(\theta_t)^{-1}\nabla f(\theta_t) \\ (FS):\ \theta_{t+1} &= \theta_t + I(\theta_t)^{-1}\nabla f(\theta_t) \end{aligned}$$

In class we'll use the demo code in the Unit 11 code file (not shown here) to try out Fisher scoring in the weight loss example.

The Gauss-Newton algorithm for nonlinear least squares involves using the FI in place of the Hessian in determining a Newton-like step. *nls()* in R uses this approach. Note that this is not exactly the same updating as our manual coding of FS for the weight loss example.

Connections between statistical uncertainty and ill-conditionedness When either the observed or expected FI matrix is nearly singular this means we have a small eigenvalue in the inverse covariance (the precision), which means a large eigenvalue in the covariance matrix. This indicates some linear combination of the parameters has low precision (high variance), and that in that direction the likelihood is nearly flat. As we've seen with N-R, convergence slows with shallow gradients, and we may have numerical problems in determining good optimization steps when the likelihood is sufficiently flat. So convergence problems and statistical uncertainty go hand in hand. One, but not the only, example of this occurs when we have nearly collinear regressors.

5.4 IRLS (IWLS) for GLMs

As most of you know, iterative reweighted least squares (also called iterative weighted least squares) is the standard method for estimation with GLMs. It involves linearizing the model and using working weights and working variances and solving a weighted least squares (WLS) problem (the generic WLS solution is $\hat{\beta} = (X^\top WX)^{-1}X^\top WY$).

Exponential families can be expressed as

$$f(y;\theta,\phi) = \exp((y\theta - b(\theta))/a(\phi) + c(y,\phi)),$$

with $E(Y) = b'(\theta)$ and Var$(Y) = b''(\theta)$. If we have a GLM in the canonical parameterization (log link for Poisson data, logit for binomial), we have the natural parameter θ equal to the linear