Optimization

To *optimize* (or *minimize*) $f(\mathbf{x}) : \mathbf{R}^n \to \mathbf{R}$ is to find $\mathbf{x}_0 \in \mathbf{R}^n$ so that $f(\mathbf{x}_0) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \mathbf{R}^n$.

Algorithms

- Use calculus to find an exact solution if you can.
- *Golden section search* does not require derivatives. See `goldenSectionSearch.R` to graph and minimize $f(x) = \frac{1}{10}x^2 - 2\sin(x)$ over $(0, 4)$. (Also try $(-15, 15)$.)
- *Gradient descent* requires first partial deriviatives.

  Recall from calculus that for a function $y = f(x_1, \ldots, x_n)$, the the *gradient* of f is defined as $\nabla f(x_1, \ldots, x_n) = \left(\frac{\partial y}{\partial x_1}, \ldots, \frac{\partial y}{\partial x_n}\right)$. To minimize f by *gradient descent*, choose an initial point $(x_1, \ldots, x_n)$ and iteratively move opposite the gradient by iterating on

  $$\mathbf{x}_{i+1} = \mathbf{x}_i - \gamma \nabla f(\mathbf{x}_i)$$

  where γ is the *step size* parameter. Note that γ can be adjusted as the algorithm proceeds; a *line search* can be used to guarantee convergence for a well-behaved f. e.g. See `gradientDescent.R`.
- *Newton's method* requires first and second partial derivatives. It is an iterative method for approximating the roots of a function, finding x such that $f(x) = 0$. In optimization, Newton's method is applied to the derivative function $f'(x)$ to find x such that $f'(x) = 0$, since such an x is a minimum, maximum, or inflection point. e.g. For the $n = 1$ case, see `Newton.R`. (The $n > 1$ case requires a *Hessian matrix* that isn't introduced in the prerequisites to STAT 305.)
- Nelder-Mead is a heuristic method that does not require derivatives. It evaluates f over a simplex, a set of $n + 1$ vertices in n dimensions that is a generalization of a triangle ($2 + 1$ vertices in 2 dimensions), repeatedly replacing the worst vertex with one computed from the others. See `NelderMead.R`.

R functions

- $n = 1$

  `optimize(f, interval, ...)` minimizes the continuous function `f`, whose opposite `-f` is unimodal, over its first argument over the interval (`interval[1]`, `interval[2]`), where `...` are additional arguments passed to `f`. It returns a list containing:

  - `minimum`, the argument that minimizes `f`
  - `objective`, the value `f(minimum)`

  `?optimize` says it uses golden section search (with successive parabolic interpolation).

  Note: If `f` is not unimodal, `optimize` may get stuck at a local minimum.

- $n \geq 1$

  `optim(par, fn, gr=NULL, ..., method=c("Nelder-Mead","BFGS","CG","L-BFGS-B","SANN","Brent")`
  minimizes the function `fn` over its first vector argument, a vector of parameters, starting at initial values in the vector `par`, where `...` are additional arguments passed to `fn`. `gr`, the gradient of `fn`, is required for some values of `method`. `optim()` returns a list containing:

  - `par`, the parameters that minimize `fn`
  - `value`, which is `f(par)`
  - `convergence`, a code with `0` indicating success, `1` indicating an iteration limit was reached, and other values indicating other trouble

  Regarding `method`,

  - `"Nelder-Mead"`, the default, does not require `gr`.
  - `"BFGS"` approximates Netwon's method; it requires `gr`
  (- `"CG"` uses a *conjugate gradient* method that may be more fragile than `"BFGS"` but useful for large problems; it requires `gr`
  - `"L-BFGS-B"` is a *limited-memory* variant of `"BFGS"` that is suitable for large n
  - `"SANN"` uses *simulated annealing*, a probabilistic heuristic for finding an approximately optimal solution; it does not require derivatives
  - `"Brent"`, useful only for $n = 1$, just calls `optimize`; it does not use `gr`
)

Statistics uses optimization to estimate parameters

Optimization finds those parameter values that make the observed data most likely.

e.g. To find the optimal simple linear regression model of the form $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$ for the data $\{(x_i, y_i) : i = 1, \cdots, n\}$, we find $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize the sum of squared errors,

$$SSE = SSE\left(\hat{\beta}_0, \hat{\beta}_1; \{(x_i, y_i)\}\right) = \sum e_i^2 = \sum (y_i - \hat{y_i})^2 = \sum \left(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)\right)^2$$

We could use optimization to find the least-squares model, but we already have a closed form solution from calculus in `lm()`. In the homework, you'll use optimization to solve an important variant that, instead of using least squares, uses least absolute deviations. e.g.

```
SSE = function(beta, x, y) { # usual least squares
  return(sum((y - (beta[1] + beta[2] * x))^2))
}
out = optim(par=c(0, 0), fn=SSE, x=mtcars$wt, y=mtcars$mpg)
m = lm(mpg ~ wt, data=mtcars)
out2 = optim(par=c(0, 0), fn=SSE, x=mtcars$wt, y=mtcars$mpg, control=list(reltol=1e-12))
# Try constant model too.
```

Optimization is a big field. See `http://cran.r-project.org/web/views/Optimization.html` for much more.