Optimization

Simon Wood, University of Edinburgh, U.K.

Slides work best with Adobe Reader

Optimization

- Optimization is concerned with finding the maximum or minimum of a function.
- For example, consider an *objective function* $D(\boldsymbol{\theta})$. We might want to find the value of parameter vector, $\boldsymbol{\theta}$, minimizing D:

$$\hat{\boldsymbol{\theta}} = \underset{\theta}{\operatorname{argmin}}\, D(\boldsymbol{\theta})$$

- Note: minimization of a function D is equivalent to maximization of $-D$, so it suffices to consider minimization.
- Many statistical modelling methods and machine learning algorithms rely on optimization, so it is important to know something about programming such methods.

Smooth optimization

- Optimization is easiest if the objective function is sufficiently smooth, and unimodal. Here, we'll assume it is.
- Then the conditions for a minimum are that

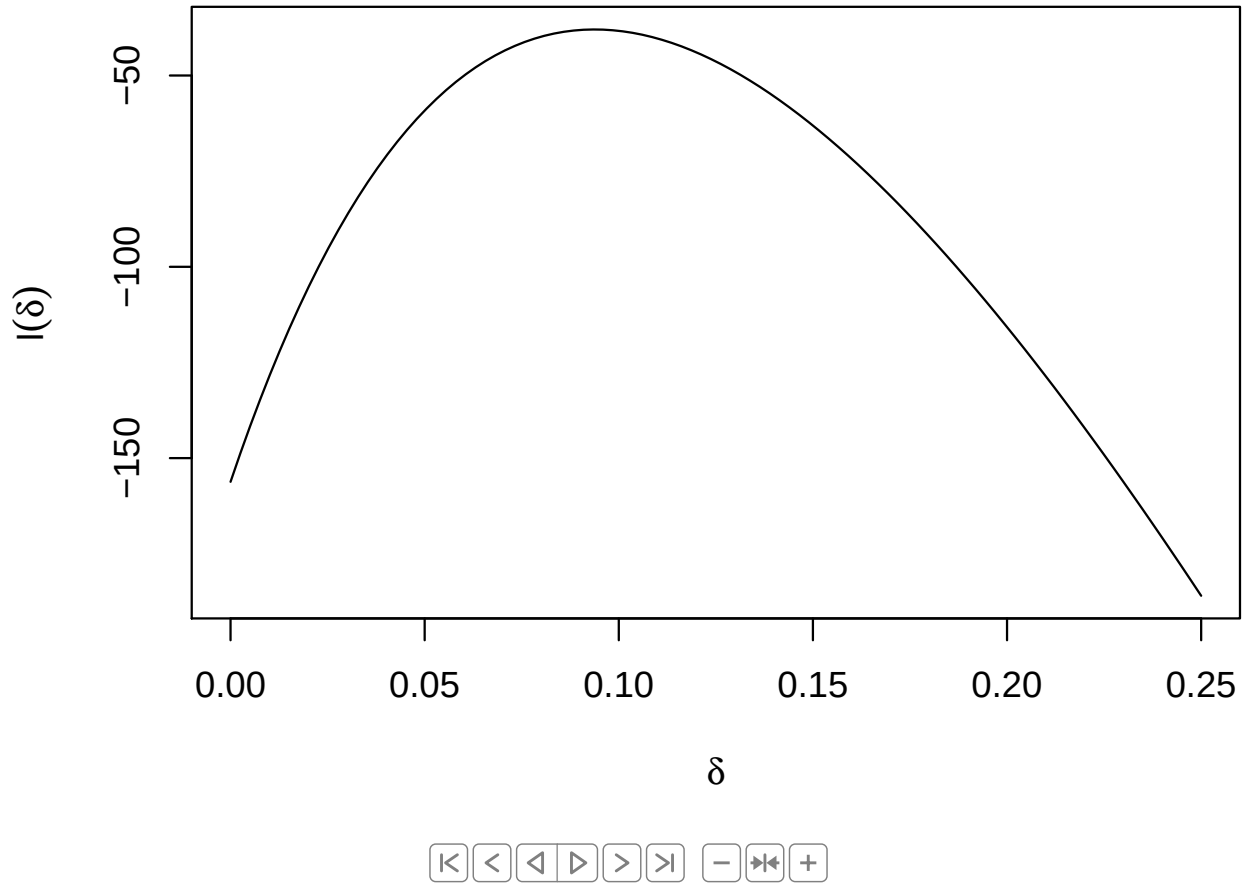
$$\nabla D = \begin{bmatrix} \frac{\partial D}{\partial \theta_1} \\ \frac{\partial D}{\partial \theta_2} \\ \cdot \\ \cdot \end{bmatrix}_{\hat{\boldsymbol{\theta}}} = \mathbf{0} \;\&\; \nabla^2 D = \begin{bmatrix} \frac{\partial^2 D}{\partial \theta_1^2} & \frac{\partial^2 D}{\partial \theta_1 \partial \theta_2} & \cdot & \cdot \\ \frac{\partial^2 D}{\partial \theta_1 \partial \theta_2} & \frac{\partial^2 D}{\partial \theta_2^2} & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}_{\hat{\boldsymbol{\theta}}} + \text{def.}$$

  ∇D is the *gradient vector*, and $\nabla^2 D$ the *Hessian matrix* of D.
- The condition on the Hessian ensures that D increases in every direction from $\hat{\boldsymbol{\theta}}$ (consider a Taylor expansion about $\hat{\boldsymbol{\theta}}$).
- Conditions for a maximum are similar, except that we require $-\nabla^2 D$ to be positive definite.

Newton's method: basic idea

- Newton's method is used to optimize smooth objective functions, such as (negative) log likelihoods, w.r.t. some parameters.
- We start with a guess of the parameter values.
- Then evaluate the function and its first and second derivatives w.r.t. the parameters at the guess.
- There is a unique quadratic function matching the value and derivatives, so we find that and optimize it to find the next guess at the optimizer of the objective.
- This derivative – quadratic approximation – maximize quadratic cycle is repeated to convergence.
- Convergence when the first derivatives are approximately zero.
- Note that Newton's method uses the same quantities that appear in the large sample results used with MLE.

Newton's method illustrated in one dimension



Newton's method in more detail

- Consider minimizing $D(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$. Taylor's theorem says

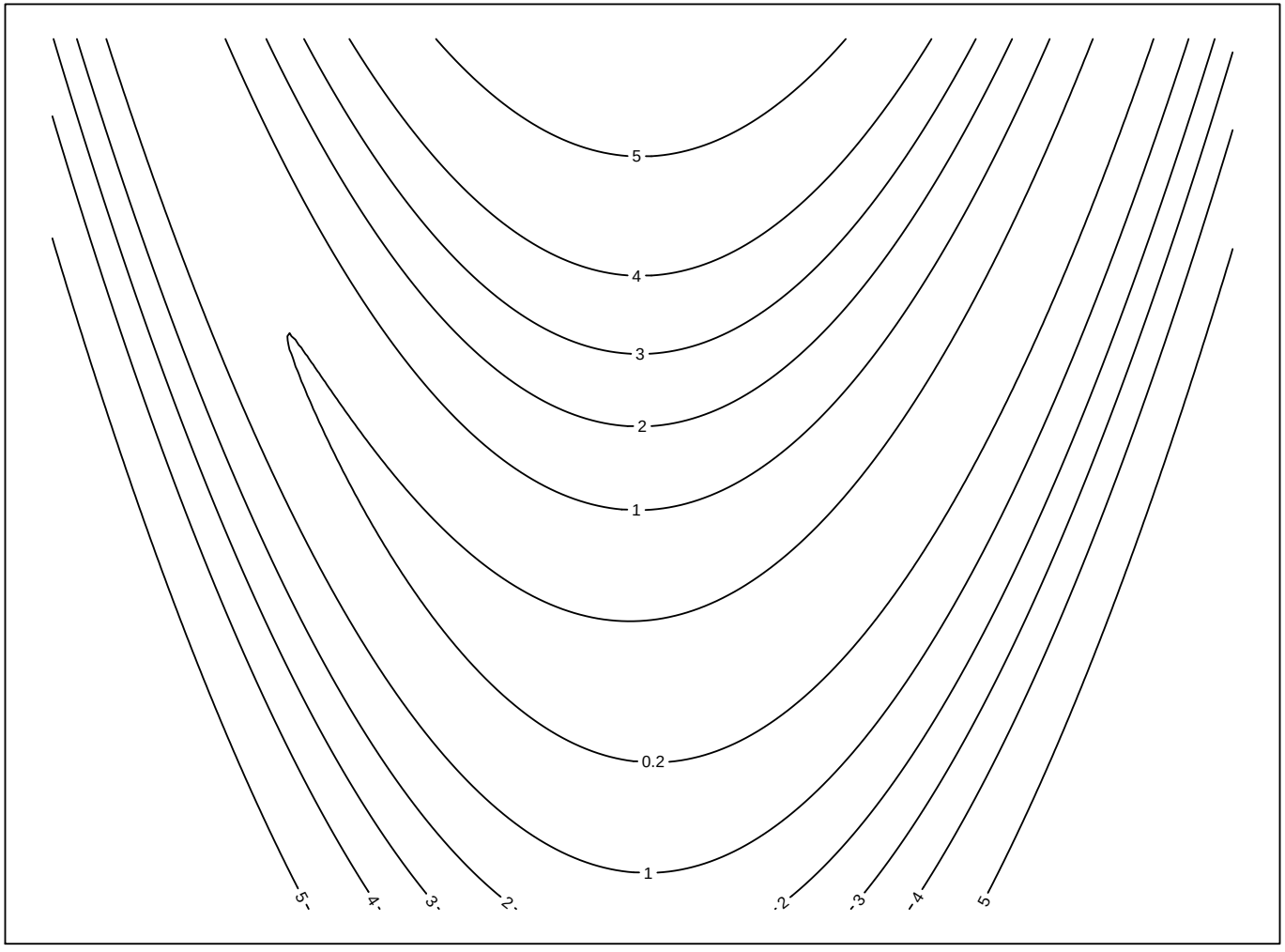
$$D(\boldsymbol{\theta} + \boldsymbol{\Delta}) = D(\boldsymbol{\theta}) + \boldsymbol{\Delta}^{\mathsf{T}} \nabla_{\theta} D + \tfrac{1}{2} \boldsymbol{\Delta}^{\mathsf{T}} \nabla_{\theta}^2 D \boldsymbol{\Delta} + o(\|\boldsymbol{\Delta}\|^2)$$

- Provided $\nabla_{\theta}^2 D$ is positive definite, the $\boldsymbol{\Delta}$ minimizing the quadratic on the right is

$$\boldsymbol{\Delta} = -(\nabla_{\theta}^2 D)^{-1} \nabla_{\theta} D$$

- This also minimizes D in the small $\boldsymbol{\Delta}$ limit, which is the one that applies near D's minimum.
- Interestingly, $\boldsymbol{\Delta}$ is a descent direction for *any positive definite matrix* in place of the Hessian $\nabla_{\theta}^2 D$.
- So if $\nabla_{\theta}^2 D$ is not positive definite we just perturb it to be so.
- Far from the optimum, $\boldsymbol{\Delta}$ might overshoot. If so, repeatedly halve $\boldsymbol{\Delta}$ until $D(\boldsymbol{\theta} + \boldsymbol{\Delta}) < D(\boldsymbol{\theta})$ to guarantee convergence,

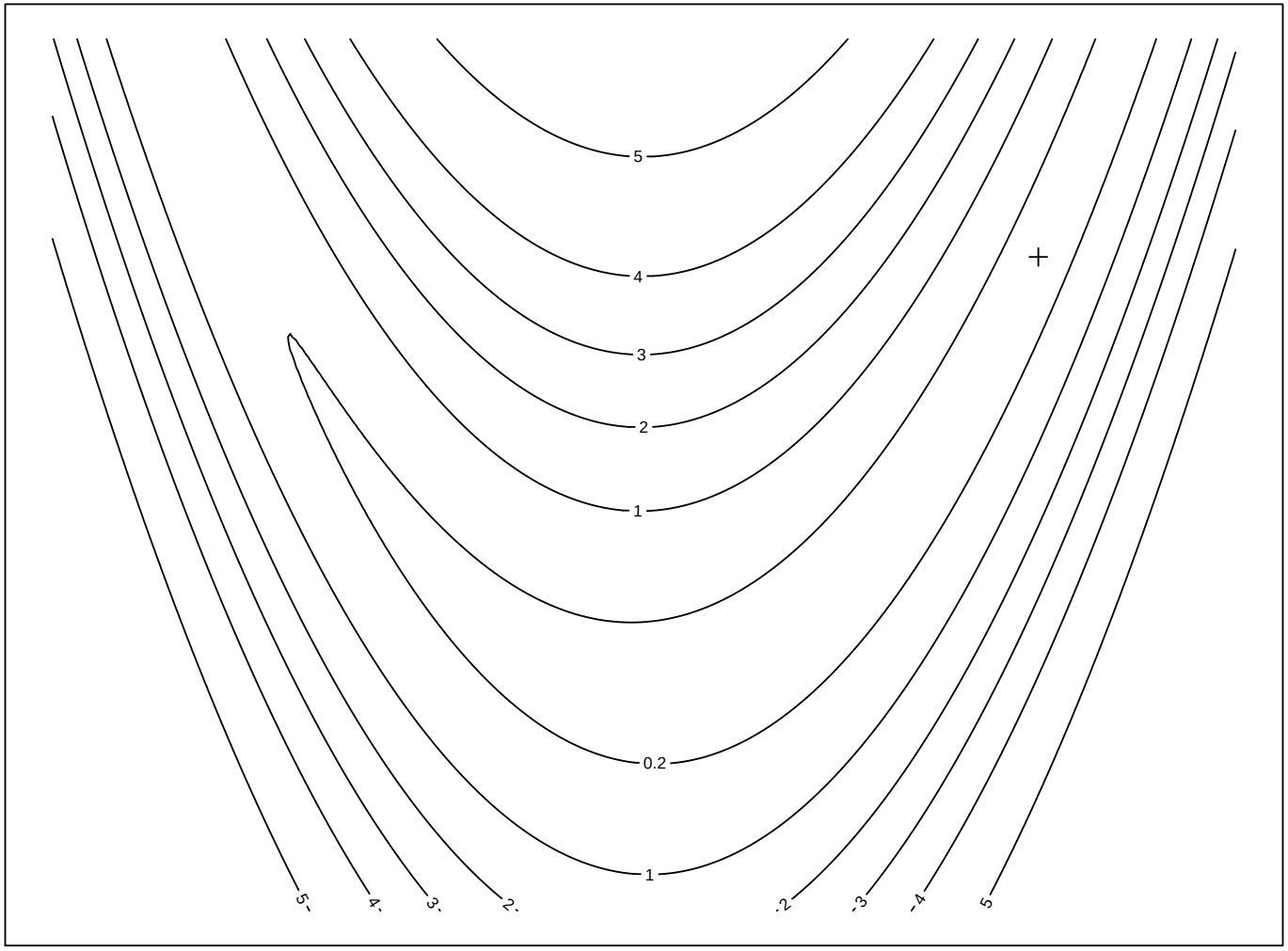
Newton in 2D with Hessian perturbation and step halving



Why Newton?

- Why not not simplify and use a first order Taylor expansion in place of the Newton method's second order expansion?
- Doing so gives the method of *steepest descent* and two problems
  1. As we approach the optimum the first derivative of the objective vanishes, so that there is ever less justification for dropping the second derivative term.
  2. Without second derivative information we have nothing to say how long the step should be.
- In practice 1. leads to steepest descent often requiring huge numbers of steps as the optimum is approached.
- For some problems co-ordinate descent can work well.
  1. Cycle through optimizing with respect to each parameter θ_i in turn, keeping the other parameters at their last updated value.
  2. The one dimensional optimizations can be cheap and easy to code, but you may need lots of them.
  3. For the previous example, Newton takes 20 steps and co-ordinate descent over 4000 (for reduced accuracy). Here are the first 20...

First 20 coordinate descent steps



Quasi Newton

- What can be done without having to calculate Hessians?
- Recall that we can use *any* positive definite matrix in place of the Hessian in the Newton update and still get a descent direction.
- So we could use an approximate Hessian.
- What if we try to build an approximate Hessian by using the second derivative information contained in the way the first derivative changes as the optimization progresses?
- In particular, after a step, update the approximate Hessian so that the quadratic model matches the gradient vector at both ends of the step.
- . . . that's the Quasi-Newton idea.
- The best known version is the BFGS* method.

---

*Broyden, Fletcher, Goldfarb, Shanno

BFGS

- Let $\boldsymbol{\theta}^{[k]}$ denote the k^{th} trial $\boldsymbol{\theta}$, with approx. inverse Hessian $\mathbf{B}^{[k]}$.
- Let $\mathbf{s}_k = \boldsymbol{\theta}^{[k+1]} - \boldsymbol{\theta}^{[k]}$ and $\mathbf{y}_k = \nabla D(\boldsymbol{\theta}^{[k+1]}) - \nabla D(\boldsymbol{\theta}^{[k]})$.
- Defining $\rho_k^{-1} = \mathbf{s}_k^\mathsf{T}\mathbf{y}_k$ the BFGS update is

$$\mathbf{B}^{[k+1]} = (\mathbf{I} - \rho_k \mathbf{s}_k \mathbf{y}_k^\mathsf{T})\mathbf{B}^{[k]}(\mathbf{I} - \rho_k \mathbf{y}_k \mathbf{s}_k^\mathsf{T}) + \rho_k \mathbf{s}_k \mathbf{s}_k^\mathsf{T}$$

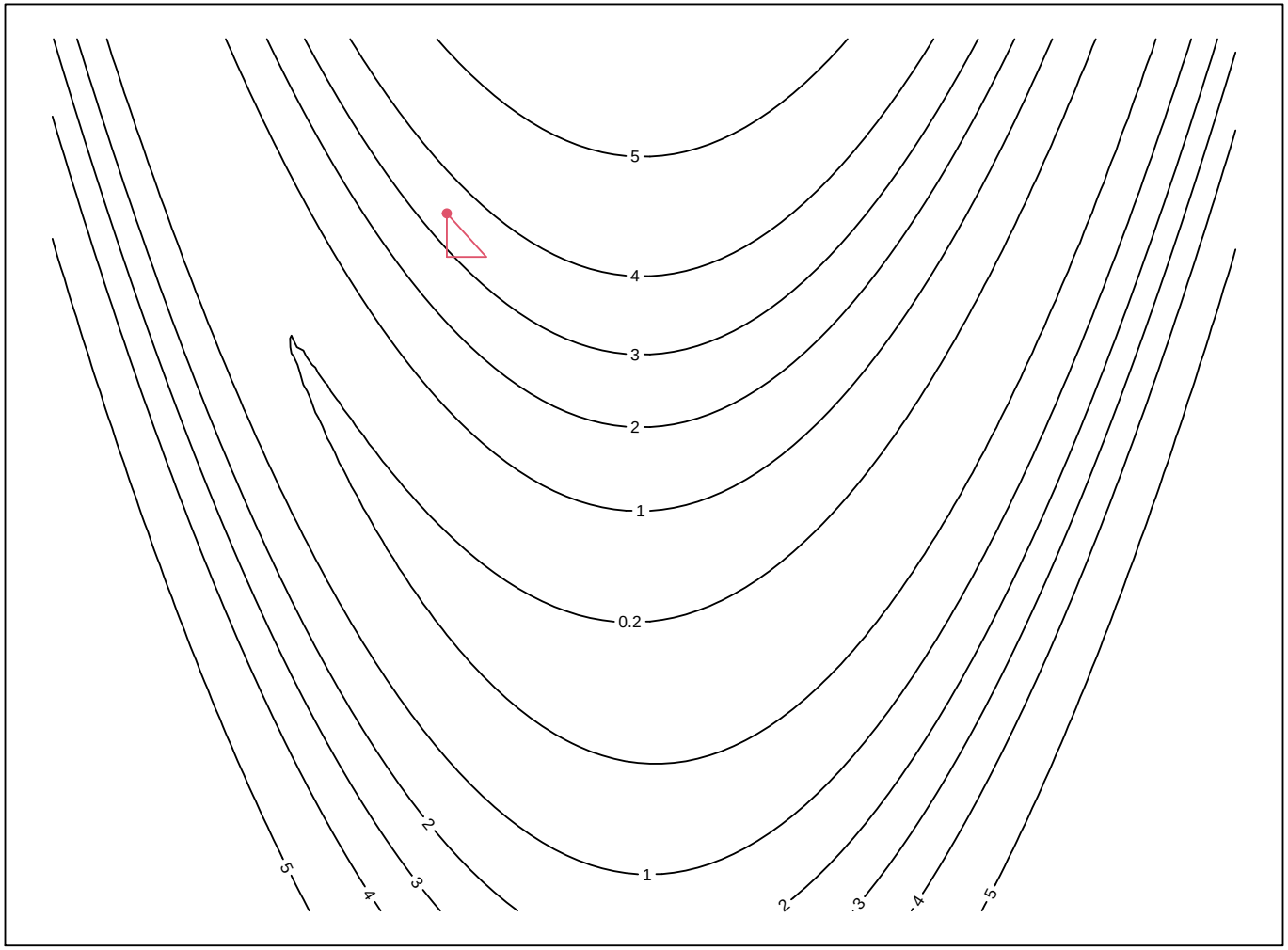
- The Quasi-Newton step from $\boldsymbol{\theta}^{[k]}$ is $\boldsymbol{\Delta} = -\mathbf{B}^{[k]}\nabla D(\boldsymbol{\theta}^{[k]})$.
- The actual step length is chosen to satisfy the Wolfe conditions
  1. $D(\boldsymbol{\theta}^{[k]} + \boldsymbol{\Delta}) \leq D(\boldsymbol{\theta}^{[k]}) + c_1 \nabla D(\boldsymbol{\theta}^{[k]})^\mathsf{T}\boldsymbol{\Delta}$
  2. $\nabla D(\boldsymbol{\theta}^{[k]} + \boldsymbol{\Delta})^\mathsf{T}\boldsymbol{\Delta} \geq c_2 \nabla D(\boldsymbol{\theta}^{[k]})^\mathsf{T}\boldsymbol{\Delta}$

  where $0 < c_1 < c_2 < 1$. Curvature condition (2) ensures that $\rho_k > 0$ so that $\mathbf{B}^{[k+1]}$ stays positive definite.
- $\mathbf{B}^{[0]}$ is sometimes set to $\mathbf{I}$, or to the inverse of a finite difference approximation to the Hessian.

Derivative free optimization: Nelder Mead

- What can be done with only function evaluations?
- Let $p = \dim(\boldsymbol{\theta})$. Define a *polytope* of $p + 1$ distinct $\boldsymbol{\theta}$ values.
- Iterate...
  1. The vector from the worst $\boldsymbol{\theta}$ point through the centroid (mean) of the other p points is the search direction.
  2. Initial step length is twice distance, d, from worst point to centroid of others. If new point is not worst one, try $3d$, picking the best of the two to replace the worst point.
  3. If previous step did not succeed, try steps of $0.5d$ and $1.5d$.
  4. If previous 2 steps failed, then linearly rescale the polytope towards the best point.

Nelder Mead method



Optimization in R

- R also has built in optimizer functions, and add on packages providing more.
- Here we will look at `optim` and `nlm`.
- `optim` offers Nelder-Mead (default), BFGS and other optimization methods.
  - the user supplies an objective function, and, optionally, a function for evaluating the gradient vector of the objective.
- `nlm` performs Newton optimization.
  - optionally the user supplied objective function can return its value with gradient vector and Hessian matrix as attributes
- Both functions will use approximate numerical derivatives when these are not supplied by the user.

optim

- `optim(par,fn,gr=NULL,...,method="Nelder-Mead")`
  - `par` is the vector of initial values for the optimization parameters.
  - `fn` is the objective function to minimize. Its first argument is always the vector of optimization parameters. Other arguments must be named, and will be passed to `fn` via the `'...'` argument to `optim`. It returns the value of the objective.
  - `gr` is as `fn`, but, if supplied, returns the gradient vector of the objective.
  - `'...'` is used to pass named arguments to `fn` and `gr`.
  - `method` selects the optimization method. `"BFGS"` is another possibility.
- Example with Rosenbrock's function

```
rb0 <- function(theta,k) { ## Rosenbrock
  z <- theta[1]; x <- theta[2]
  k * (z - x^2)^2 + (1 - x)^2
} ## rb0
optim(c(-.5,1), rb0, k=10)
```

nlm

- `nlm(f, p, ...)`
  - `f` is the objective function, exactly like `fn` for `optim`. In addition its return value may optionally have `'gradient'` and `'hessian'` attributes.
  - `p` is the vector of initial values for the optimization parameters.
  - `'...'` is used to pass named arguments to `f`.
- Example with Rosenbrock's function

```
rb0 <- function(theta,k) { ## Rosenbrock
  z <- theta[1]; x <- theta[2]
  k * (z - x^2)^2 + (1 - x)^2
} ## rb0
nlm(rb0, c(-.5,1), k=10)
```

Approximating derivatives

- `optim` and `nlm` approximate any required derivatives not supplied by the user. How?
- Using *finite differencing*[†]. Let $\boldsymbol{e}_i$ be a vector of zeroes except for its i^{th} element which is one. Then for a small ϵ

$$\frac{\partial D}{\partial \theta_i} \simeq \frac{D(\boldsymbol{\theta} + \epsilon \mathbf{e}_i) - D(\boldsymbol{\theta})}{\epsilon}$$

- How small should ϵ be?
  - Too large and the approximation will be poor (Taylor's theorem).
  - Too small and $\boldsymbol{\theta} + \epsilon \mathbf{e}_i$ will only differ from $\boldsymbol{\theta}$ in a few (or even none!) of its least significant digits, losing precision.
  - For many well scaled problems $\epsilon \simeq \sqrt{\epsilon_{\text{machine}}}$, where $\epsilon_{\text{machine}}$ is the smallest value for which $1 + \epsilon_{\text{machine}}$ does not have the same floating point representation as 1: the *machine precision*.

[†] See *Core Statistics*, §5.5.2

Numerically exact derivatives

- We can work out the derivatives of the objective by hand, or using a computer algebra package, and code them up.
  - **Always** check such derivative code by finite differencing.
  - **Always, Always, Always**.
- Alternatively we can use *automatic differentiation* (AD) methods[‡], which compute derivatives of a function directly from the code implementing the function.
- R function `deriv` offers a simple AD implementation for differentiation of R expressions.
- Let's use it to obtain a function evaluating Rosenbrock's function, its gradient vector and Hessian matrix (attached as attributes to the return value).

```
rb <- deriv(expression(k*(z-x^2)^2 + (1-x)^2),
            c("x","z"), ## diff w.r.t. these
            function.arg=c("x","z","k"),
            hessian=TRUE)
```

[‡]See *Core Statistics* §5.5.3