Chapter 12 - Optimisation

- Newton's method for optimisation
- The golden-section method
- Multivariate optimisation
- Steepest ascent
- Newton's method in higher dimensions
- A curve-fitting example

Suppose $f : \mathbb{R} \to \mathbb{R}$ with continuous first and second derivatives:

- f has a *global maximum* at x^* if $f(x) \leq f(x^*)$ for all x.
- f has a *local maximum* at x^* if $f(x) \leq f(x^*)$ for all x in a neighbourhood of x^*.

Conditions for local maximum:

- Necessary condition for x^* to be a local maximum is $f'(x^*) = 0$ and $f''(x^*) \leq 0$.
- Sufficient condition is $f'(x^*) = 0$ and $f''(x^*) < 0$.

A *local search* technique generates a sequence of points, $x(0)$, $x(1)$, $x(2)$, ..., which (hopefully) converge to a local maximum of f.

- Given a prospective solution $x(n)$, we look for the next prospective solution $x(n+1)$ in some neighbourhood of $x(n)$.

- Because they never consider the whole space of possible solutions, local search techniques can only ever be guaranteed to find local maxima.
- Let x^* be a local maximum of f. Supposing $x(n) \longrightarrow x^*$ as $n \longrightarrow \infty$, we need *stopping criteria* to decide when to stop searching:
  - $|x(n) - x(n-1)| \leq \varepsilon$;
  - $|f(x(n)) - f(x(n-1))| \leq \varepsilon$;
  - $|f'(x(n))| \leq \varepsilon$.
- Local search techniques may not converge at all.
  - For example if f is unbounded then $x(n) \longrightarrow \infty$.
  - Usually specify a maximum number of iterations $n_{\max}$.

12.1 - Newton's method for optimisation

If $f : [a,b] \to \mathbb{R}$ has a continuous derivative f', then the maximum of f is the maximum of

- $f(a)$, $f(b)$, and
- $f(x_1), \ldots, f(x_n)$, where $x_1, \ldots, x_n$ are the roots of f'.

If we apply the Newton-Raphson method for root-finding to f', we get the Newton method for optimising f:

$$x(n+1) = x(n) - \frac{f'(x(n))}{f''(x(n))}.$$

```
newton <- function(f3, x0, tol = 1e-9, n.max = 100) {
    # Newton's method for optimisation, starting at x0
    # f3 is a function that given x returns the vector
    # (f(x), f'(x), f''(x)), for some f
    x <- x0
    f3.x <- f3(x)
    n <- 0
    while ((abs(f3.x[2]) > tol) & (n < n.max)) {
        x <- x - f3.x[2]/f3.x[3]
        f3.x <- f3(x)
        n <- n + 1
    }
    if (n == n.max) { cat('newton failed to converge\n')}
       else { return(x)}
}
newton(gamma.2.3, 0.25)
```

12.1 - Newton's method for optimisation (cont.)

- When the Newton algorithm converges, we can end up with a minimum or a maximum since all such stationary points satisfy $f'(x^*) = 0$.
- Because we are searching for a point x^* such that $f'(x^*) = 0$, we will use $|f'(x(n))| < \varepsilon$ as our stopping condition.
- Provided $x(0)$ is close to x^*, $x(n) \longrightarrow x^*$ quickly, as $n \longrightarrow \infty$.
- We revisit Newton's method later, on a higher plane (that is, in higher dimensions).

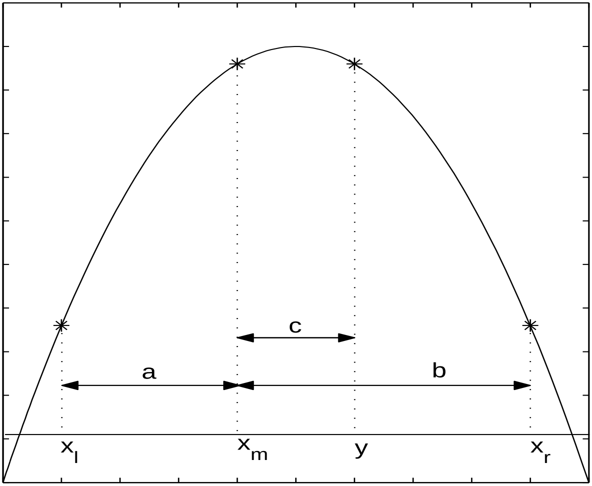
Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous function.

The golden-section method works in one dimension only, but does not need f'.

The golden-section method is similar to the root-bracketing technique for root-finding.

- To determine if we have a local maximum we need three points: if $x_l < x_m < x_r$ and $f(x_l) \leq f(x_m)$ and $f(x_m) \leq f(x_r)$ then there must be a local maximum in the interval $[a, b]$.

12.2 - The golden-section method



How to choose y?:

- Suppose that (x_m, x_r) is the larger interval
- let $a = x_m - x_l$, $b = x_r - x_m$, and $c = y - x_m$.
- The golden-section algorithm chooses y so that the ratio of the lengths of the larger to the smaller interval stays the same at each iteration.
- That is, if the new bracketing interval is $[x_l, y]$ then

$$\frac{a}{c} = \frac{b}{a}$$

while if the new bracketing interval is $[x_m, x_r]$ then

$$\frac{b-c}{c} = \frac{b}{a}.$$

12.2 - The golden-section method: Algorithm

- Put $\rho = b/a$ then solving these for c we get

$$\rho^2 - \rho - 1 = 0 \quad \text{so} \quad \rho = \frac{1+\sqrt{5}}{2}$$

  which is the famous golden ratio. We also get $a = b - c$, so $c = b/(1+\rho)$ and thus $y = x_m + c = x_m + (x_r - x_m)/(1+\rho)$.
- The length ratio of the new interval to the old is either $b/(a+b)$ or $(a+c)/(a+b)$, which both work out as $\rho/(1+\rho)$.
- An analogous argument applies if (x_l, x_m) is the larger interval.

The argument above shows that if we start with x_m chosen so that the ratio $(x_r - x_m)/(x_m - x_l) = \rho$ or $1/\rho$, then at each iteration the width of the bracketing interval is reduced by a factor of $\rho/(1+\rho)$ and so must eventually go to zero.

Golden-section method 2 Start with $x_l < x_m < x_r$ such that $f(x_l) \leq f(x_m)$ and $f(x_r) \leq f(x_m)$

1 if $x_r - x_l \leq \varepsilon$ then stop

2 if $x_r - x_m > x_m - x_l$ then do 2a otherwise do 2b

  2a let $y = x_m + (x_r - x_m)/(1+\rho)$
  if $f(y) \geq f(x_m)$ then put $x_l = x_m$ and $x_m = y$ otherwise put $x_r = y$

  2b let $y = x_m - (x_m - x_l)/(1+\rho)$
  if $f(y) \geq f(x_m)$ then put $x_r = x_m$ and $x_m = y$ otherwise put $x_l = y$

3 go back to step 1

```
gsection <- function(ftn, x.l, x.r, x.m, tol = 1e-9) {
  # applies the golden-section algorithm to maximise ftn
  # we assume that ftn is a function of a single variable
  # and that x.l < x.m < x.r and ftn(x.l), ftn(x.r) <= ftn(x.m)
  #
  # the algorithm iteratively refines x.l, x.r, and x.m and terminates
  # when x.r - x.l <= tol, then returns x.m

  # golden ratio plus one
  gr1 <- 1 + (1 + sqrt(5))/2

  return(x.m)
}

f <- function(x) ifelse (x==0, 0, abs(x)*log(abs(x)/2)*exp(-abs(x)))
curve(f, -10, 10, n = 501)
gsection(f, 1, 5, 2)
gsection(f, -1, 1, .1)
```

Let $f : \mathbb{R}^d \to \mathbb{R}$ and suppose that all of the first- and second-order partial derivatives of f exist and are continuous everywhere.

- We write $\mathbf{x} = (x_1, \ldots, x_d)^T$ for an element of $\mathbb{R}^d$ and
- $\mathbf{e}_i$ for the i-th co-ordinate vector: $\mathbf{x} = x_1\mathbf{e}_1 + \cdots + x_d\mathbf{e}_d$.
- The i-th partial derivative at $\mathbf{x}$ will be denoted $f_i(\mathbf{x}) = \partial f(\mathbf{x})/\partial x_i$ and we define the *gradient*

$$\nabla f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_d(\mathbf{x}))^T$$

and the *Hessian*

$$\mathbf{H}(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_d \partial x_1} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_d \partial x_d} \end{pmatrix}.$$

- The slope at $\mathbf{x}$ in direction $\mathbf{v} \neq \mathbf{0}$ is given by

$$\mathbf{v}^T \nabla f(\mathbf{x}) / \|\mathbf{v}\|,$$

  where $\|\mathbf{v}\| = \sqrt{v_1^2 + \cdots + v_d^2}$ is the Euclidean norm.
- The curvature at $\mathbf{x}$ in direction $\mathbf{v}$ is given by

$$\mathbf{v}^T \mathbf{H}(\mathbf{x}) \mathbf{v} / \|\mathbf{v}\|^2.$$

- f has a local maximum at $\mathbf{x}$ if for all $\varepsilon > 0$ small enough, $f(\mathbf{x} + \varepsilon \mathbf{e}_i) \leq f(\mathbf{x})$ for $i = 1, \ldots, d$.

12.3 - Multivariate Optimisation

- f has a local maximum at $\mathbf{x}$ if for all $\varepsilon > 0$ small enough, $f(\mathbf{x} + \varepsilon \mathbf{e}_i) \leq f(\mathbf{x})$ for $i = 1, \ldots, d$.
- A necessary (but not sufficient) condition for a local maximum at $\mathbf{x}$ is:
  - $\nabla f(\mathbf{x}) = \mathbf{0} = (0, \ldots, 0)^T$ and for all $\mathbf{v} \neq \mathbf{0}$
  - the curvature at $\mathbf{x}$ in direction $\mathbf{v}$ is ≤ 0 (we say that the Hessian is *negative semi-definite*).
- A sufficient (but not necessary) condition for f to have a local maximum at $\mathbf{x}$ is that:
  - $\nabla f(\mathbf{x}) = \mathbf{0}$ and
  - the curvature in all directions is < 0 (in which case we say that the Hessian $\mathbf{H}(\mathbf{x})$ is *negative-definite*).

12.3 - Multivariate Optimisation

- As in one dimension, we will use iterative local search techniques to find local maxima.
- Define $\|\mathbf{x}\|_\infty = \max_i |x_i|$ (the L_∞ norm).
- In higher dimensions we use stopping conditions that are combinations of the following:
  - $\|\mathbf{x}(n) - \mathbf{x}(n-1)\|_\infty \leq \varepsilon$;
  - $|f(\mathbf{x}(n)) - f(\mathbf{x}(n-1))| \leq \varepsilon$;
  - $\|\nabla f(\mathbf{x}(n))\|_\infty \leq \varepsilon$.
- To guard against non-convergence, we should also specify a maximum number of iterations $n_{\max}$, then stop when $n = n_{\max}$.

Put $\mathbf{x}(n+1) = \mathbf{x}(n) + \alpha\mathbf{v}$, where α is a positive scalar and the direction $\mathbf{v}$ is the direction with largest slope.

At point $\mathbf{x}$, the direction with largest slope is $\nabla f(\mathbf{x})$.

Thus, the steepest method has the form

$$\mathbf{x}(n+1) = \mathbf{x}(n) + \alpha\nabla f(\mathbf{x}(n)),$$

for some $\alpha \geq 0$.

12.4 - Steepest Ascent

Given $\mathbf{x}(n+1) = \mathbf{x}(n) + \alpha \nabla f(\mathbf{x}(n))$, we choose $\alpha \geq 0$ to maximise

$$g(\alpha) = f\left(\mathbf{x}(n) + \alpha \nabla f(\mathbf{x}(n))\right).$$

- If $\alpha = 0$ then we have reached a local maximum.
- If $\alpha > 0$ then $f(\mathbf{x}(n+1)) > f(\mathbf{x}(n))$.

If f is bounded above then, because $f(\mathbf{x}(n+1)) \geq f(\mathbf{x}(n))$, the sequence $\{f(\mathbf{x}(n))\}_{n=1}^{\infty}$ must converge. This suggests that we can use the stopping condition:

- $f(\mathbf{x}(n)) - f(\mathbf{x}(n-1)) \leq \varepsilon$, for some small tolerance ε.

It can be shown that if f is bounded and ∇f is 'well behaved', then the sequence $\{\mathbf{x}(n)\}_{n=1}^{\infty}$ will converge to a local maximum.

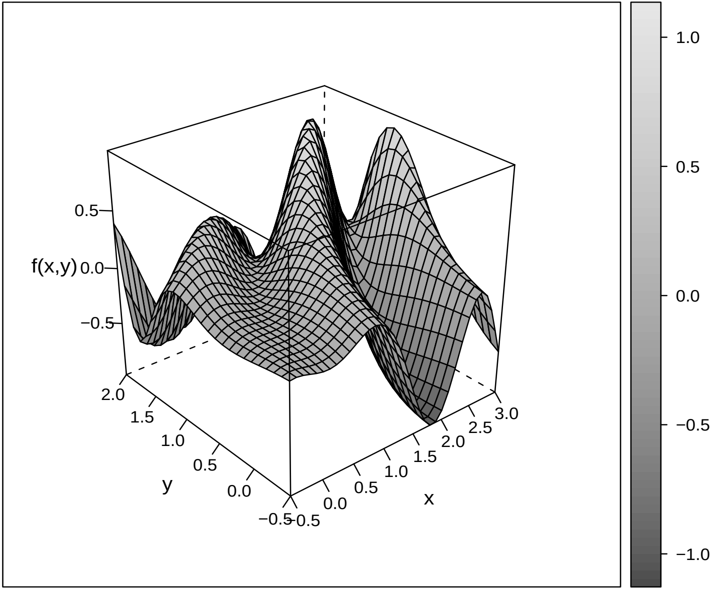
```r
ascent <- function(f, grad.f, x0, tol = 1e-9, n.max = 100) {
    # steepest ascent algorithm
    # find a local max of f starting at x0
    # function grad.f is the gradient of f

    x.old <- x0
    x <- line.search(f, x0, grad.f(x0))
    n <- 1
    while ((f(x) - f(x.old) > tol) & (n < n.max)) {
        x.old <- x
        x <- line.search(f, x, grad.f(x))
        n <- n + 1
    }
    return(x)
}
```

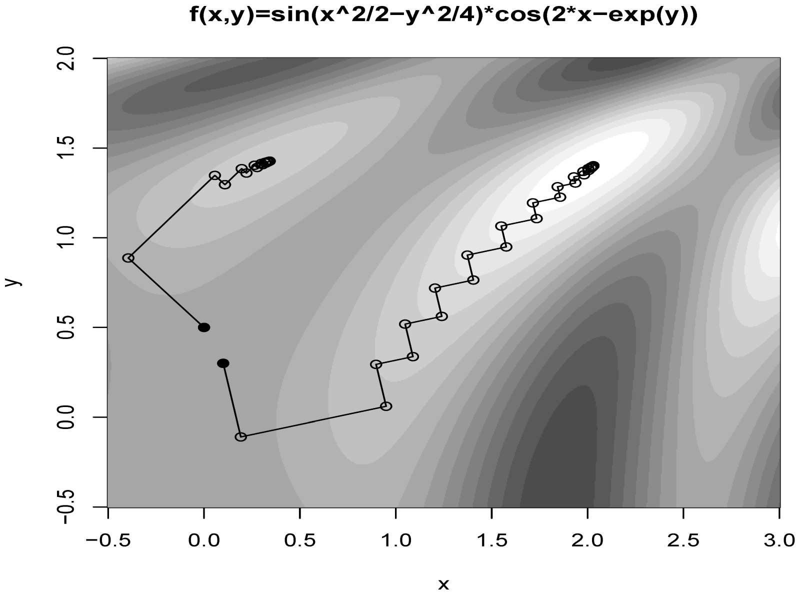
12.4.1 - Line Search

To maximise $g(\alpha) = f(\mathbf{x}(n) + \alpha \nabla f(\mathbf{x}(n)))$ over $\alpha \geq 0$, golden-section algorithm will be used.

We require three initial points $\alpha_l < \alpha_m < \alpha_r$ such that $g(\alpha_m) \geq g(\alpha_l)$ and $g(\alpha_m) \geq g(\alpha_r)$:

- Put $\alpha_l = 0$.
- In theory, if $\|\nabla f(\mathbf{x}(n))\| > 0$ then $g'(0) > 0$ and thus there must be some $\varepsilon > 0$ such that $g(\varepsilon) > g(0)$, so we can put $\alpha_m = \varepsilon$.
- Unfortunately there is not even a theoretical guarantee that a suitable α_r exists, because we may have g increasing over the whole interval $[0, \infty)$.
  Hence, we specify a *maximum step size* $\alpha_{\max}$ and if we cannot find $\alpha_r \leq \alpha_{\max}$ such that $g(\alpha_r) \leq g(\alpha_m)$, we just return $\alpha_{\max}$.


0.5
f(x,y) 0.0
−0.5
2.0
1.5
1.0
0.5
y
0.0
−0.5
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x
1.0
0.5
0.0
−0.5
−1.0


f(x,y)=sin(x^2/2−y^2/4)*cos(2*x−exp(y))
2.0
1.5
1.0
0.5
0.0
−0.5
y
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x

12.5 Newton's method in higher dimensions

The basis of the method is a second-order Taylor expansion of f. For any $\mathbf{x}$ and $\mathbf{y}$ close together we have

$$f(\mathbf{y}) \approx f(\mathbf{x}) + (\mathbf{y}-\mathbf{x})^T \nabla f(\mathbf{x}) + \frac{1}{2}(\mathbf{y}-\mathbf{x})^T \mathbf{H}(\mathbf{x})(\mathbf{y}-\mathbf{x}).$$

Taking partial derivatives w.r.t. $\mathbf{y}$ we get

$$\nabla f(\mathbf{y}) \approx \nabla f(\mathbf{x}) + \mathbf{H}(\mathbf{x})(\mathbf{y}-\mathbf{x}).$$

If $\mathbf{y}$ is a local maximum then $\nabla f(\mathbf{y}) = \mathbf{0}$ and, solving the equation above, we get

$$\mathbf{y} = \mathbf{x} - \mathbf{H}(\mathbf{x})^{-1} \nabla f(\mathbf{x}).$$

Suppose $\mathbf{x}(n)$ is our current estimate, then we would like our next estimate $\mathbf{x}(n+1)$ to be a local maximum (at least approximately) ...

Newton's algorithm

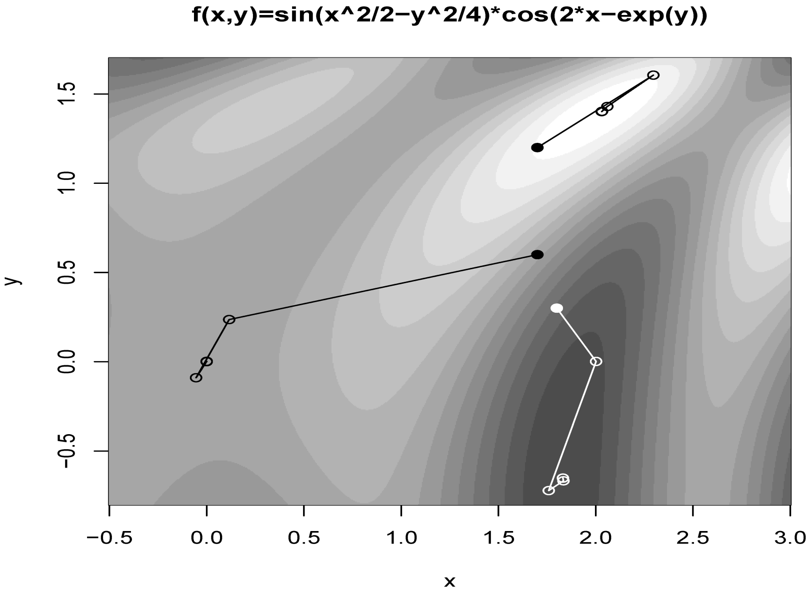
$$\mathbf{x}(n+1) = \mathbf{x}(n) - \mathbf{H}(\mathbf{x}(n))^{-1}\nabla f(\mathbf{x}(n)).$$

- Clearly if $\mathbf{H}(\mathbf{x}(n))$ is singular (has no inverse), then Newton's method breaks down.
- Even if $\mathbf{H}(\mathbf{x}(n))$ is non-singular at each step, Newton's method may not converge.
- Despite this, if f has a local maximum at $\mathbf{x}^*$, f is 'nicely behaved' near $\mathbf{x}^*$, and if our initial point $\mathbf{x}(0)$ is 'close enough' to $\mathbf{x}^*$, then Newton's method will converge to $\mathbf{x}^*$ quickly.

12.5 - Newton's method in higher dimensions

- In implementing Newton's method we will assume that we have some function **`f3`** that takes argument **`x`** and returns a list containing $f(x)$, $\nabla f(x)$, and $\mathbf{H}(x)$.
- For our stopping condition we will use $\|\nabla f(\mathbf{x}(n))\|_\infty \leq \varepsilon$.

```r
newton <- function(f3, x0, tol = 1e-9, n.max = 100) {
  # f3 returns the list {f(x), grad f(x), Hessian f(x)}, for some f
  x <- x0
  f3.x <- f3(x)
  n <- 0
  while ((max(abs(f3.x[[2]])) > tol) & (n < n.max)) {
    x <- x - solve(f3.x[[3]], f3.x[[2]])
    f3.x <- f3(x)
    n <- n + 1
  }
  if (n == n.max) { cat('newton failed to converge\n') }
    else { return(x)}
```


f(x,y)=sin(x^2/2−y^2/4)*cos(2*x−exp(y))
1.5
1.0
0.5
0.0
−0.5
y
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x

12.5 - Newton's method in higher dimensions

- Newton's method needs to calculate the gradient and Hessian.
- Steepest ascent only requires the gradient, but sometimes even this can be difficult.

If ∇f is unavailable then there are two approaches we can take:

- The first assumes that even if we don't know what they are, $\mathbf{H}$ and/or ∇f do exist, in which case we can try and estimate them.
- The second approach is to use an optimisation method that does not require the gradient.
  - Such approaches tend to be relatively slow, but relatively reliable. In one dimension the golden-section algorithm is an example of a derivative-free approach.
  - In higher dimensions there is an algorithm due to Nelder & Mead, which is well accepted and again is derivative-free.

Suppose we have observations $(x_1, y_1), \ldots, (x_n, y_n)$ and we want to find a function f such that $y_i \approx f(x_i)$ for $i = 1, \ldots, n$.

Further suppose that f can be *parameterised* by some vector of parameters $\theta = (\theta_1, \ldots, \theta_d)^T$. For example, if we restrict f to be a quadratic then it has the form $f(x) = ax^2 + bx + c$, in which case $\theta = (a, b, c)^T$.

We write $f(x;\theta)$ for $f(x)$ to emphasise the dependence on θ.

The problem of finding the parameter θ^*, such that the fitted points $\hat{y}_i = f(x_i; \theta^*)$ are 'closest' to the observations y_i, is called *curve fitting*.

Curve fitting

To measure how close the fitted points are to the observed points, we use a *loss function*. Two popular choices are the sum of squares

$$L_2(\theta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

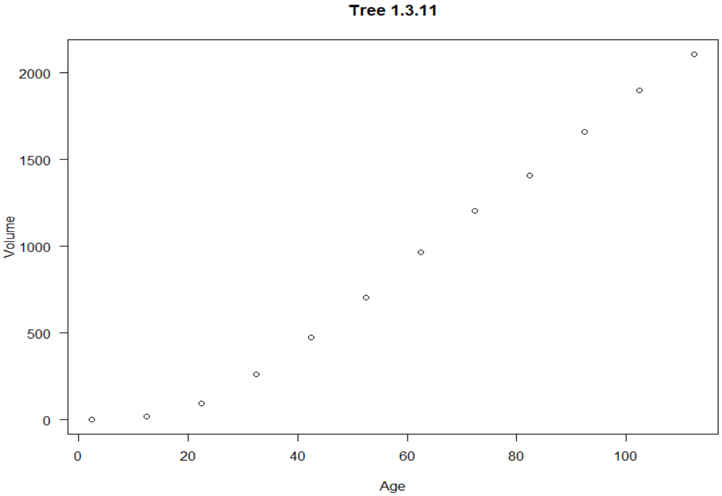
and the sum of absolute differences

$$L_1(\theta) = \sum_{i=1}^{n} |y_i - \hat{y}_i|.$$

Note that we consider a loss function to be a function of θ, rather than a function of $\mathbf{y}$, because we are interested in how the loss changes as we change θ.

Given a loss function L, we choose θ^* to be that θ that minimises $L(\theta)$.

Curve fitting



```
richards <- function(t, theta)
  theta[1]*(1 - exp(-theta[2]*t))^theta[3]

loss.L2 <- function(theta, age, vol)
  sum((vol - richards(age, theta))^2)

loss.L1 <- function(theta, age, vol)
  sum(abs(vol - richards(age, theta)))

trees <- read.csv("../data/trees.csv")
tree <- trees[trees$ID=="1.3.11", 2:3]

theta0 <- c(1000, 0.1, 3)
theta.L2 <- optim(theta0, loss.L2, age=tree$Age, vol=tree$Vol)
theta.L1 <- optim(theta0, loss.L1, age=tree$Age, vol=tree$Vol)
```

```
plot(tree$Age, tree$Vol, type="p", xlab="Age", ylab="Volume",
     main="Tree 1.3.11")
lines(tree$Age, richards(tree$Age, theta.L2$par), col="blue")
lines(tree$Age, richards(tree$Age, theta.L1$par), col="blue", lty=2)

attach(tree)
fit <- nls(Vol ~ a*(1 - exp(-b*Age))^c, start=list(a=1000, b=0.01,
     c=3))
summary(fit)
```