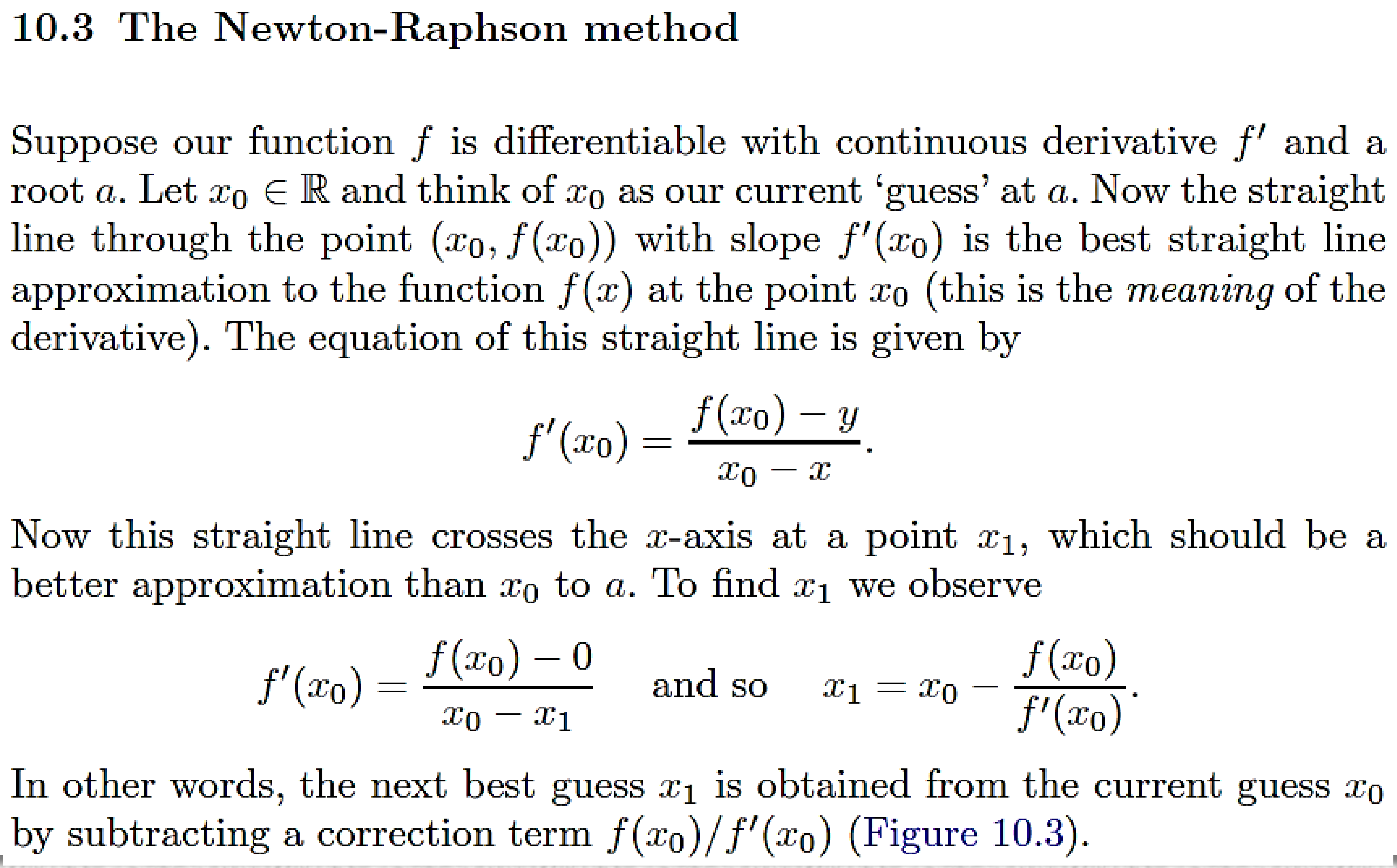
Programming for Simulation and MC Methods

Root-finding

Newton-Raphson



10.3 The Newton-Raphson method

Suppose our function f is differentiable with continuous derivative f' and a root a. Let $x_0 \in \mathbb{R}$ and think of x_0 as our current 'guess' at a. Now the straight line through the point $(x_0, f(x_0))$ with slope $f'(x_0)$ is the best straight line approximation to the function $f(x)$ at the point x_0 (this is the *meaning* of the derivative). The equation of this straight line is given by

$$f'(x_0) = \frac{f(x_0) - y}{x_0 - x}.$$

Now this straight line crosses the x-axis at a point x_1, which should be a better approximation than x_0 to a. To find x_1 we observe

$$f'(x_0) = \frac{f(x_0) - 0}{x_0 - x_1} \quad \text{and so} \quad x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

In other words, the next best guess x_1 is obtained from the current guess x_0 by subtracting a correction term $f(x_0)/f'(x_0)$ (Figure 10.3).

Newton-Raphson

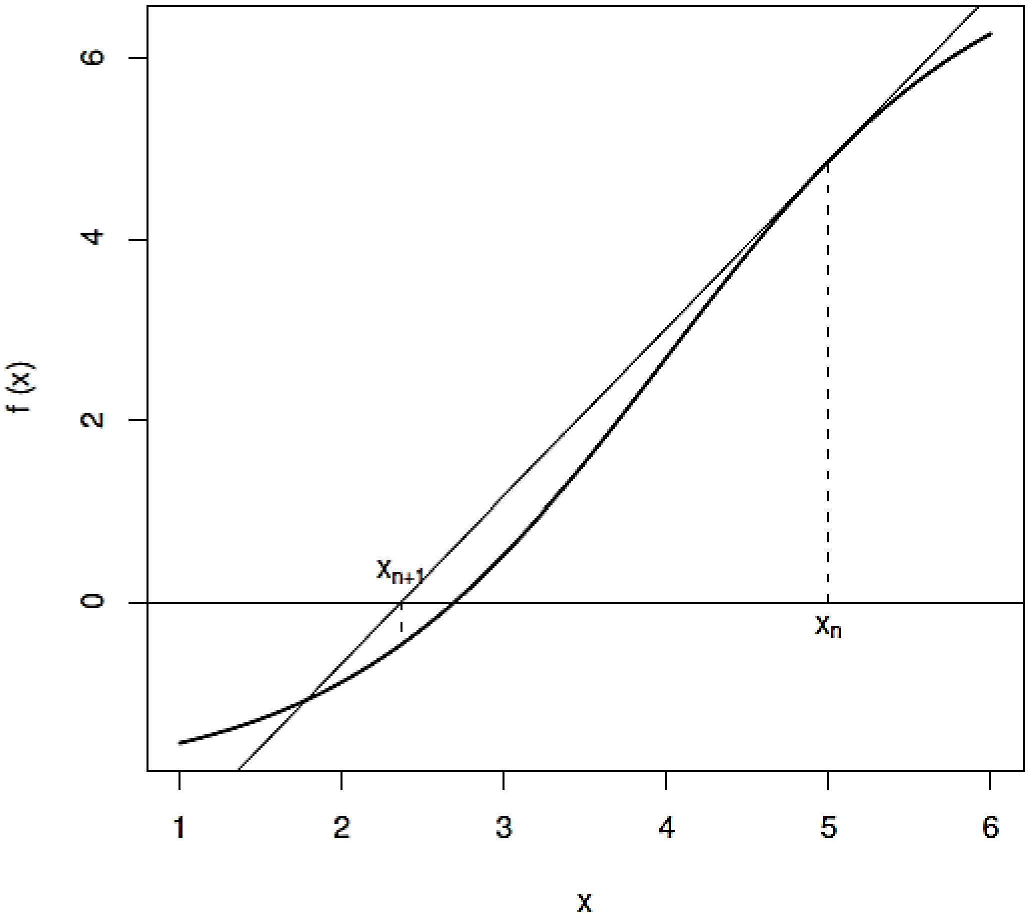


Figure 10.3 *A step in the Newton-Raphson root-finding method.*

Newton-Raphson

Now that we have x_1, we use the same procedure to get the next guess

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

or in general:

Newton-Raphson method

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

Newton-Raphson

Newton-Raphson method

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

Like the fixed-point method, this is a first-order recurrence relation. It can be shown that if f is 'well behaved' at a (which means $f'(a) \neq 0$ and f'' is finite and continuous at a)[1] and you start with x_0 'close enough' to a, then x_n will converge to a quickly. Unfortunately, like the fixed-point method, we don't know if f is well behaved at a until we know a, and we don't know beforehand how close is close enough.

So, we cannot guarantee convergence of the Newton-Raphson algorithm. However, if $x_n \to a$ then, since f and f' are continuous, we have

$$a = \lim_{n\to\infty} x_{n+1} \quad = \quad \lim_{n\to\infty} \left(x_n - \frac{f(x_n)}{f'(x_n)} \right)$$

Newton-Raphson

$$= \lim_{n\to\infty} x_n - \frac{f(\lim_{n\to\infty} x_n)}{f'(\lim_{n\to\infty} x_n)} = a - \frac{f(a)}{f'(a)}.$$

Thus, provided $f'(a) \neq \pm\infty$, we must have $f(a) = 0$.

Since we are expecting $f(x_n) \to 0$, a good stopping condition for the Newton-Raphson algorithm is $|f(x_n)| \leq \epsilon$ for some tolerance ϵ. If the sequence $\{x_n\}_{n=0}^{\infty}$ is converging to a root a, then for x close to a we have $f(x) \approx f'(a)(x-a)$. So if $|f(x_n)| \leq \epsilon$ we have $|x-a| \leq \epsilon/f'(a)$ (approximately).

The code below implements the Newton-Raphson algorithm in a function `newtonraphson`. To use it you first need to create a function, `ftn(x)` say, which returns the vector $(f(x), f'(x))$. `newtonraphson(ftn, x0, tol = 1e-9, max.iter = 100)` has four inputs:

Newton-Raphson

`ftn` is the name of a function that takes a single numeric input and returns a numeric vector of length two. If x is the input then the output must be $(f(x), f'(x))$.

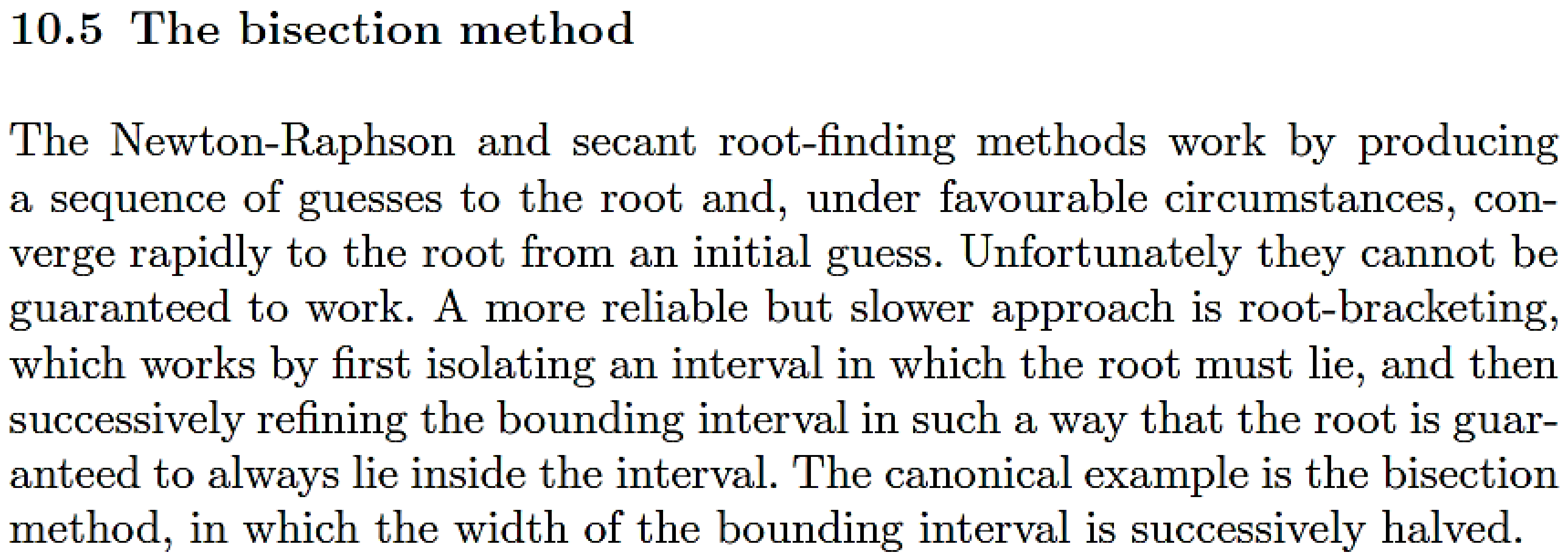
`x0` is the starting point for the algorithm.

`tol` is such that the algorithm will stop if $|f(x_n)| \leq \text{tol}$, with default 10^{-9}.

`max.iter` is such that the algorithm will stop when $n = \text{max.iter}$, with default 100.

As for the fixed-point method, because we cannot guarantee convergence, we count the number of iterations and stop if this gets too large. This prevents the program running indefinitely, though of course you have to make sure that you do not stop it too soon, in case it is converging more slowly than you expected. Note that, because our stopping condition only depends on $|f(x_n)|$, and not $|x_n - x_{n-1}|$, we do not have to store the previous iteration, as we did with function `fixedpoint`.

Bisection Method



10.5 The bisection method

The Newton-Raphson and secant root-finding methods work by producing a sequence of guesses to the root and, under favourable circumstances, converge rapidly to the root from an initial guess. Unfortunately they cannot be guaranteed to work. A more reliable but slower approach is root-bracketing, which works by first isolating an interval in which the root must lie, and then successively refining the bounding interval in such a way that the root is guaranteed to always lie inside the interval. The canonical example is the bisection method, in which the width of the bounding interval is successively halved.

Bisection Method

Suppose that f is a continuous function, then it is easy to see that f has a root in the interval (x_l, x_r) if either $f(x_l) < 0$ and $f(x_r) > 0$ or $f(x_l) > 0$ and $f(x_r) < 0$. A convenient way to verify this condition is to check if $f(x_l)f(x_r) < 0$. The bisection method works by taking an interval (x_l, x_r) that contains a root, then successively refining x_l and x_r until $x_r - x_l \leq \epsilon$, where ϵ is some predefined tolerance. The algorithm is as follows:

```
Bisection method Start with $x_l < x_r$ such that $f(x_l)f(x_r) < 0$.
1. if $x_r - x_l \leq \epsilon$ then stop.
2. put $x_m = (x_l + x_r)/2$; if $f(x_m) = 0$ then stop.
3. if $f(x_l)f(x_m) < 0$ then put $x_r = x_m$ otherwise put $x_l = x_m$.
4. go back to step 1.
```

Bisection Method

```
Bisection method Start with $x_l < x_r$ such that $f(x_l)f(x_r) < 0$.
1. if $x_r - x_l \le \epsilon$ then stop.
2. put $x_m = (x_l + x_r)/2$; if $f(x_m) = 0$ then stop.
3. if $f(x_l)f(x_m) < 0$ then put $x_r = x_m$ otherwise put $x_l = x_m$.
4. go back to step 1.
```

Note that at every iteration of the algorithm, we know that there is root in the interval (x_l, x_r). Provided we start with $f(x_l)f(x_r) < 0$, the algorithm is guaranteed to converge, with the approximation error reducing by a constant factor $^1/_2$ at each iteration. If we stop when $x_r - x_l \le \epsilon$, then we know that both x_l and x_r are within distance ϵ of a root.

Bisection Method

Note that the bisection method cannot find a root a if the function f just touches the x-axis at a, that is, if the x-axis is a tangent to the function at a. The Newton-Raphson method will still work in this case. The most popular current root-finding methods use root-bracketing to get close to a root, then switch over to the Newton-Raphson or secant method when it seems safe to do so. This strategy combines the safety of bisection with the speed of the secant method.