Numerical Integration (spuRs Ch.11)

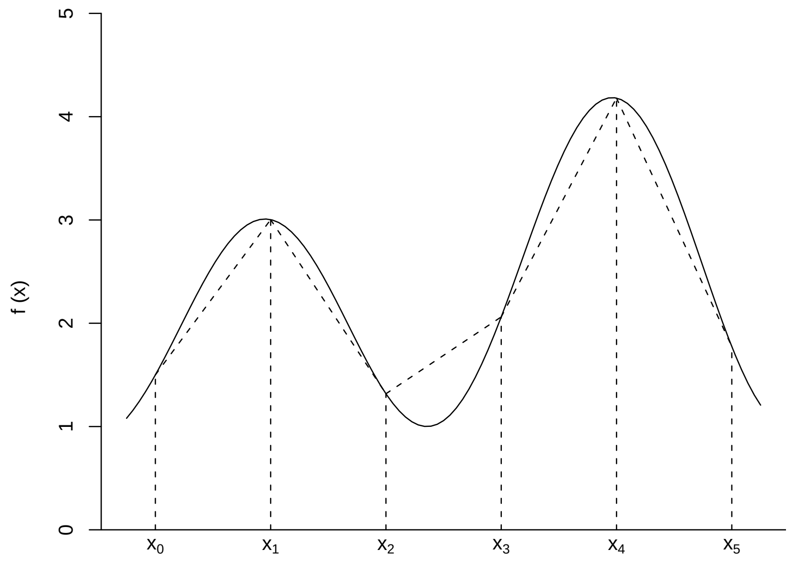
It is frequently necessary to compute definite integrals $\int_a^b f(x)dx$ of a given function f. From the Fundamental Theorem of Calculus we know that if we can find an antiderivative F, such that $F'(x) = f(x)$, then $\int_a^b f(x)dx = F(b) - F(a)$. However for many functions f it is impossible to write down an antiderivative in closed form. In such cases we can use numerical integration to approximate the definite integral.

For example, in statistics we often use definite integrals of the standard normal density, that is, integrals of the form

$$\Phi(z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\, dx.$$

We know that $\Phi(0) = 1/2$ and $\Phi(\infty) = 1$, but for all other z numerical integration is used.

Trapezoidal rule



We split $[a, b]$ into sub-intervals of width h. Let $x_0, \ldots, x_n$ be the end points of the sub-intervals. The trapezoidal approximation is obtained by approximating the area under $y = f(x)$ over the subinterval $[x_i, x_{i+1}]$ by a trapezoid.

The width of the trapezoid is h, the left side of the trapezoid has height $f(x_i)$ and the right side has height $f(x_{i+1})$. The area of the trapezoid is thus

$$\frac{h}{2}(f(x_i) + f(x_{i+1})).$$

Now we add the areas for all of the subintervals together to get our trapezoidal approximation T to the integral $\int_a^b f(x)dx$:

Trapezoidal rule

$$T = \frac{h}{2}(f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)).$$

```
source("../scripts/trapezoid.r")
ftn6 <- function(x) return(4*x^3)
trapezoid(ftn6, 0, 1, n = 20)
trapezoid(ftn6, 0, 1, n = 40)
trapezoid(ftn6, 0, 1, n = 60)
```

Note that as defined, the function `ftn6` is vectorised (given a vector as input, it will return a vector as output). Thus, in `trapezoid`, the command `sapply(x.vec, ftn)` could be replaced by `ftn(x.vec)`. The advantage of using `sapply` is that it will work even if `ftn` is not vectorised.

Simpson's rule

Simpon's rule subdivides the interval $[a, b]$ into n subintervals, where n is even, then on each consecutive pair of subintervals, it approximates the behaviour of $f(x)$ by a parabola (polynomial of degree 2) rather than by the straight lines used in the trapezoidal rule.

Let $u < v < w$ be any three points distance h apart. For $x \in [u, w]$ we want to approximate $f(x)$ by a parabola which passes through the points $(u, f(u))$, $(v, f(v))$, and $(w, f(w))$. There is exactly one such parabola $p(x)$ and it is given by the formula

$$p(x) = f(u)\frac{(x-v)(x-w)}{(u-v)(u-w)} + f(v)\frac{(x-u)(x-w)}{(v-u)(v-w)} + f(w)\frac{(x-u)(x-v)}{(w-u)(w-v)}.$$

As an approximation to the area under the curve $y = f(x)$, we use $\int_u^w p(x)dx$. A rather lengthy but elementary calculation shows

$$\int_u^w p(x)dx = \frac{h}{3}(f(u) + 4f(v) + f(w)).$$

Adding up the approximations for the subintervals $[x_{2i}, x_{2i+2}]$ we obtain

Simpson's rule

$$S = \frac{h}{3}(f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+\cdots+4f(x_{n-1})+f(x_n)).$$

```
source("../scripts/simpson_n.r")
ftn6 <- function(x) return(4*x^3)
simpson_n(ftn6, 0, 1, 20)
```

Convergence of Simpson's rule

To test the accuracy of Simpson's rule we estimated

$$\int_{0.01}^{1} (1/x)\,dx = -\log(0.01)$$

for a sequence of increasing values of n, the number of partitions. A plot of $\log(\text{error})$ against $\log(n)$ appears to have a slope of roughly -4 for large values of n, indicating that the error decays like n^{-4}. This can in fact be shown to hold in general for functions f with a continuous fourth derivative.

```
source("../scripts/simpson_test.r")
```

size of the partition

When using `simpson_n` in practice we need some rule for choosing n which results in a reasonably accurate approximation.

or for choosing h, the length of the subintervals

Suppose that f is continuous and we wish to estimate $I = \int_a^b f(x)dx$. Let $S(n)$ be the value of the approximation when we use a partition of size n, then $S(n) \to I$ as $n \to \infty$. Thus $S(2n) - S(n) \to 0$ and we can use a stopping rule of the form n large enough that $|S(2n) - S(n)| \leq \epsilon$, where $\epsilon > 0$ is some small tolerance.

As a rule of thumb, the square root of machine epsilon is a good place to start when choosing ϵ, that is, around 10^{-8} if you are working in double precision.

```
source("../scripts/simpson.r")
```

or equivalently, choose h small enough so that $|S(h/2) - S(h)|$ is less than a pre-specified tolerance value. See also adaptive quadrature.

Adaptive quadrature

In adaptive quadrature, the subinterval width h is not constant over the interval $[a, b]$, but instead adapts to the function. The key observation is that h only needs to be small where the integrand f is steep.

To see why it is a good idea to change the size of h to suit the behaviour of the function, we will consider the integral $\int_0^1 (k+1)x^k dx = 1$. How large a partition is required to estimate $\int_0^1 (k+1)x^k dx = 1$ using a tolerance of 10^{-9}?

```
source("../scripts/simpson.r")
options(digits = 16)
f4 <- function(x) 5*x^4
simpson(f4, 0, 1, tol = 1e-9, verbose = TRUE)
f8 <- function(x) 9*x^8
simpson(f8, 0, 1, tol = 1e-9, verbose = TRUE)
f12 <- function(x) 13*x^12
simpson(f12, 0, 1, tol = 1e-9, verbose = TRUE)
```

Clearly as k increases $(k+1)x^k$ gets steeper and we need a smaller h to achieve a given tolerance. But $(k+1)x^k$ is much steeper over the interval $[0.5, 1]$ than the interval $[0, 0.5]$. Thus if we split the integral into two bits, we should find that we need a much smaller h for the interval $[0.5, 1]$ than for $[0, 0.5]$:

```
S1 <- simpson(f12, 0, 0.5, tol = 0.5e-9, verbose = TRUE)
S2 <- simpson(f12, 0.5, 1, tol = 0.5e-9, verbose = TRUE)
S1 + S2
options(digits = 7)
```

Note that when we split the integral into two, we halved the tolerance for each part. That way, the tolerance for the recombined integral `S1 + S2` is guaranteed to remain less than 10^{-9}.

Adaptive quadrature automatically allows the interval width h to vary over the range of integration, using a recursive algorithm. The basic idea is to apply Simpson's rule using some initial h and $h/2$. If the difference between the two estimates is less than some given tolerance ϵ, then we are done. If not then we split the range of integration $[a, b]$ into two parts ($[a, c]$ and $[c, b]$ where $c = (a + b)/2$) and on each part we apply Simpson's rule using interval widths $h/2$ and $h/4$ and a tolerance of $\epsilon/2$.

If the desired tolerance is not met on a given subinterval then we split it further, but we only do this for subintervals that do not achieve the desired tolerance.

We apply quadrature to the function $f(x) = 1.5\sqrt{x}$ over the range [0, 1].

```
rm(list = ls())
source("../scripts/quadrature.r")
ftn <- function(x) return(1.5*sqrt(x))
quadrature(ftn, 0, 1, tol = 1e-3, trace = TRUE)
```

Using a more realistic tolerance, we compare adaptive quadrature to the standard Simpson's rule:

```
quadrature(ftn, 0, 1, 1e-9, trace = FALSE)
source("../scripts/simpson.r")
simpson(ftn, 0, 1, 1e-9, verbose = TRUE)
```

We can also use the `system.time` function to compare the efficiency of the algorithms, and see that adaptive quadrature is substantially faster:

```
rm(list = ls())
ftn <- function(x) return(1.5*sqrt(x))
source("../scripts/quadrature.r")
system.time(quadrature(ftn, 0, 1, 1e-9, trace = FALSE))
source("../scripts/simpson.r")
system.time(simpson(ftn, 0, 1, 1e-9, verbose = FALSE))
```