3.7 R SUPPLEMENT

3.7.1 Plotting functions in various ways

Using `curve`:

Plot a Michaelis-Menten curve:

```
> curve(2 * x/(1 + x))
```

You do need to specify the parameters: if you haven't defined `a` and `b` previously `curve(a*x/(b+x))` will give you an error. But if you're going to use a function a lot it can be helpful to define a function:

```
> micmen <- function(x, a = 2, b = 1) {
+     a * x/(b + x)
+ }
```

Now plot several curves (being more specific about the desired x and y ranges; changing colors; and adding a horizontal line (`abline(h=...)`) to show the asymptote).

```
> curve(micmen(x), from = 0, to = 8, ylim = c(0, 10))
> curve(micmen(x, b = 3), add = TRUE, col = 2)
> curve(micmen(x, a = 8), add = TRUE, col = 3)
> abline(h = 8)
```

Sometimes you may want to do things more manually. Use `seq` to define x values:

```
> xvec <- seq(0, 10, by = 0.1)
```

Then use vectorization (`yvec=micmen(xvec)`) or `sapply` (`yvec=sapply(xvec,micmen)`) or a `for` loop (`for i in (1:length(xvec)) { yvec[i]=micmen(xvec[i])}`) to calculate the y values. Use `plot(xvec,yvec,...)`, `lines(xvec,yvec,...)`, etc. (with options you learned Chapter 2) to produce the graphics.

3.7.2 Piecewise functions using `ifelse`

The `ifelse` function picks one of two numbers (or values from one of two vectors) depending on a logical condition. For example, a simple threshold function:

```
> curve(ifelse(x < 5, 1, 2), from = 0, to = 10)
```

or a piecewise linear function:

```
> curve(ifelse(x < 5, 1 + x, 6 - 3 * (x - 5)), from = 0,
+     to = 10)
```

You can also nest `ifelse` functions to get more than one switching point:

```
> curve(ifelse(x < 5, 1 + x, ifelse(x < 8, 6 - 3 *
+     (x - 5), -3 + 2 * (x - 8))), from = 0, to = 10)
```

3.7.3 Derivatives

You can use `D` or `deriv` to calculate derivatives (although R will not simplify the results at all): `D` gives you a relatively simple answer, while `deriv` gives you a function that will compute the function and its derivative for specified values of x (you need to use `attr(...,"grad")` to retrieve the derivative — see below). To use either of these functions, you need to use `expression` to stop R from trying to interpret the formula.

```
> D(expression(log(x)), "x")
```

```
1/x
```

```
> D(expression(x^2), "x")
```

```
2 * x
```

```
> logist <- expression(exp(x)/(1 + exp(x)))
> dfun <- deriv(logist, "x", function.arg = TRUE)
```

```
> xvec <- seq(-4, 4, length = 40)
> y <- dfun(xvec)
> plot(xvec, y)
> lines(xvec, attr(y, "grad"))
```

Use `eval` to fill in parameter values:

```
> d1 <- D(expression(a * x/(b + x)), "x")
> d1
```

```
a/(b + x) - a * x/(b + x)^2
```

```
> eval(d1, list(a = 2, b = 1, x = 3))
```

```
[1] 0.125
```