Circle 4

Over-Vectorizing

We skirted past Plutus, the fierce wolf with a swollen face, down into the fourth Circle. Here we found the lustful.

It is a good thing to want to vectorize when there is no effective way to do so. It is a bad thing to attempt it anyway.

A common reflex is to use a function in the apply family. This is not vectorization, it is loop-hiding. The `apply` function has a `for` loop in its definition. The `lapply` function buries the loop, but execution times tend to be roughly equal to an explicit `for` loop. (Confusion over this is understandable, as there is a significant difference in execution speed with at least some versions of S+.) Table 4.1 summarizes the uses of the apply family of functions.

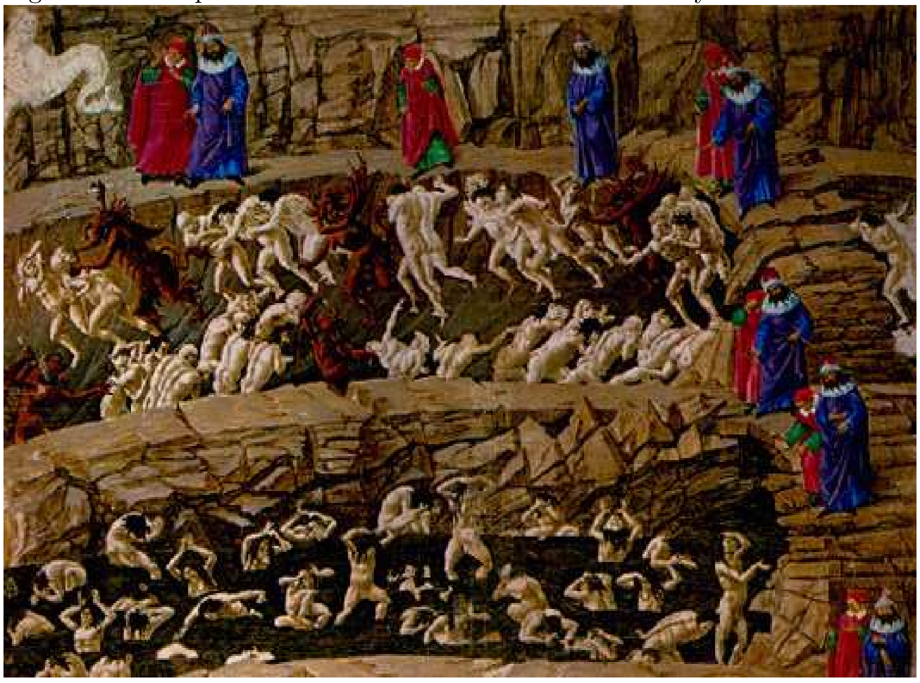
Base your decision of using an apply function on Uwe's Maxim (page 20). The issue is of human time rather than silicon chip time. Human time can be wasted by taking longer to write the code, and (often much more importantly) by taking more time to understand subsequently what it does.

A command applying a complicated function is unlikely to pass the test.

Table 4.1: The apply family of functions.

function	**input**	**output**	**comment**
`apply`	matrix or array	vector or array or list	
`lapply`	list or vector	list	
`sapply`	list or vector	vector or matrix or list	simplify
`vapply`	list or vector	vector or matrix or list	safer simplify
`tapply`	data, categories	array or list	ragged
`mapply`	lists and/or vectors	vector or matrix or list	multiple
`rapply`	list	vector or list	recursive
`eapply`	environment	list	
`dendrapply`	dendogram	dendogram	
`rollapply`	data	similar to input	package `zoo`

Figure 4.1: The panderers and seducers and the flatterers by Sandro Botticelli.



Use an explicit `for` loop when each iteration is a non-trivial task. But a simple loop can be more clearly and compactly expressed using an apply function.

There is at least one exception to this rule. We will see in Circle 8.1.56 that if the result will be a list and some of the components can be `NULL`, then a `for` loop is trouble (big trouble) and `lapply` gives the expected answer.

The `tapply` function applies a function to each bit of a partition of the data. Alternatives to `tapply` are `by` for data frames, and `aggregate` for time series and data frames. If you have a substantial amount of data and speed is an issue, then `data.table` may be a good solution.

Another approach to over-vectorizing is to use too much memory in the process. The `outer` function is a wonderful mechanism to vectorize some problems. It is also subject to using a lot of memory in the process.

Suppose that we want to find all of the sets of three positive integers that sum to 6, where the order matters. (This is related to partitions in number theory.) We can use `outer` and `which`:

```
the.seq <- 1:4
which(outer(outer(the.seq, the.seq, '+'), the.seq, '+') == 6,
      arr.ind=TRUE)
```

This command is nicely vectorized, and a reasonable solution to this particular problem. However, with larger problems this could easily eat all memory on a machine.

Suppose we have a data frame and we want to change the missing values to zero. Then we can do that in a perfectly vectorized manner:

```
x[is.na(x)] <- 0
```

But if x is large, then this may take a lot of memory. If—as is common—the number of rows is much larger than the number of columns, then a more memory efficient method is:

```
for(i in 1:ncol(x)) x[is.na(x[,i]), i] <- 0
```

Note that "large" is a relative term; it is usefully relative to the amount of available memory on your machine. Also note that memory efficiency can also be time efficiency if the inefficient approach causes swapping.

One more comment: if you really want to change `NA`s to 0, perhaps you should rethink what you are doing—the new data are fictional.

It is not unusual for there to be a tradeoff between space and time.

Beware the dangers of premature optimization of your code. Your first duty is to create clear, correct code. Only consider optimizing your code when:

- Your code is debugged and stable.
- Optimization is likely to make a significant impact. Spending an hour or two to save a millisecond a month is not best practice.