Using The `iterators` Package

Rich Calaway
richcalaway@revolution-computing.com

June 26, 2009

1 Introduction

An *iterator* is a special type of object that generalizes the notion of a looping variable. When passed as an argument to a function that knows what to do with it, the iterator supplies a sequence of values. The iterator also maintains information about its state, in particular its current index. The `iterators` package includes a number of functions for creating iterators, the simplest of which is `iter`, which takes virtually any R object and turns it into an iterator object. The simplest function that operates on iterators is the `nextElem` function, which when given an iterator, returns the next value of the iterator. For example, here we create an iterator object from the sequence 1 to 10, and then use `nextElem` to iterate through the values:

```
> library(iterators)
> i1 <- iter(1:10)
> nextElem(i1)
```

```
[1] 1
```

```
> nextElem(i1)
```

```
[1] 2
```

You can create iterators from matrices and data frames, using the `by` argument to specify whether to iterate by row or column:

```
> istate <- iter(state.x77, by = "row")
> nextElem(istate)
```

```
        Population Income Illiteracy Life Exp Murder HS Grad Frost  Area
Alabama       3615   3624        2.1    69.05   15.1    41.3    20 50708
```

```
> nextElem(istate)
```

```
       Population Income Illiteracy Life Exp Murder HS Grad Frost   Area
Alaska        365   6315        1.5    69.31   11.3    66.7   152 566432
```

Iterators can also be created from functions, in which case the iterator can be an endless source of values:

```
> ifun <- iter(function() sample(0:9, 4, replace = TRUE))
> nextElem(ifun)
```

```
[1] 5 7 7 7
```

```
> nextElem(ifun)
```

```
[1] 2 1 9 7
```

For practical applications, iterators can be paired with `foreach` to obtain parallel results quite easily:

```
> library(foreach)
```

```
foreach: simple, scalable parallel programming from REvolution Computing
Use REvolution R for scalability, fault tolerance and more.
http://www.revolution-computing.com
```

```
> x <- matrix(rnorm(1e+06), ncol = 10000)
> itx <- iter(x, by = "row")
> foreach(i = itx, .combine = c) %dopar% mean(i)
```

```
  [1] -1.399021e-02  6.703415e-03 -3.902093e-03 -6.871640e-03 -2.531504e-02
  [6] -8.963886e-03  8.117389e-03 -5.734292e-03  3.941437e-03  7.931396e-04
 [11]  4.901059e-03 -4.512940e-03 -1.411464e-02 -4.988578e-03  5.038828e-03
 [16] -6.341762e-04 -4.130914e-03  6.843467e-03  8.106298e-03 -8.510878e-03
 [21]  6.376870e-03 -4.525190e-03 -2.991466e-03  3.422304e-03 -1.022700e-02
 [26] -9.486196e-03  5.651183e-03  1.019357e-02 -5.343451e-03 -1.606776e-03
```

```
 [31] -1.845046e-02  4.128974e-03 -5.376407e-03  1.742136e-03  1.177739e-02
 [36]  5.652047e-03  7.365876e-03 -4.585533e-03  3.304031e-03 -6.897650e-03
 [41]  5.948317e-03  3.482671e-05  1.643158e-02 -7.316453e-03 -1.657182e-02
 [46]  1.900464e-02 -1.726968e-02  7.151540e-03  4.927544e-03  2.921504e-03
 [51]  6.279004e-03  1.144282e-02 -1.119096e-03  4.999337e-03  1.338732e-03
 [56]  6.889534e-03 -7.568093e-03 -1.953916e-02 -3.115634e-03  1.051931e-03
 [61] -8.277021e-03 -2.437974e-03 -7.501185e-04  1.220363e-02 -1.531169e-03
 [66]  1.442942e-02 -1.295408e-03  5.144023e-03  3.672619e-03 -3.657642e-03
 [71]  2.015284e-03 -9.733274e-03  8.444890e-03 -6.570395e-03 -5.036423e-03
 [76]  1.471503e-02  1.149881e-02 -6.163143e-03 -7.715976e-03 -1.725716e-02
 [81] -5.357699e-03  4.247016e-03 -1.350637e-02  5.460376e-03  1.277178e-02
 [86]  1.842567e-03 -1.515363e-03  3.283940e-03 -1.370248e-02  4.871467e-03
 [91]  1.836458e-03 -1.504736e-02  1.880172e-02 -1.403648e-02 -7.969673e-04
 [96]  6.098132e-03 -4.864967e-03  6.138976e-03  8.625218e-03  2.480979e-03
```

2 Some Special Iterators

The notion of an iterator is new to R, but should be familiar to users of languages such as Python. The `iterators` package includes a number of special functions that generate iterators for some common scenarios. For example, the `irnorm` function creates an iterator for which each value is drawn from a specified random normal distribution:

```
> library(iterators)
> itrn <- irnorm(10)
> nextElem(itrn)

 [1]  0.7204040  0.3410843  1.3736118 -1.6951802 -0.7801742 -0.5786290
 [7]  1.3866481  2.0109698 -0.5428345  0.9876990

> nextElem(itrn)

 [1] -0.1382349  0.6444322  0.2681039  0.7238490 -0.3062045  0.5468832
 [7]  0.5137709 -0.4470722 -0.7501819 -0.1722753
```

Similarly, the `irunif`, `irbinom`, and `irpois` functions create iterators which drawn their values from uniform, binomial, and Poisson distributions, respectively.

We can then use these functions just as we used `irnorm`:

```
> itru <- irunif(10)
> nextElem(itru)
```

```
 [1] 0.9606848 0.8970205 0.9307195 0.8507693 0.1220950 0.3909116 0.6796561
 [8] 0.2259454 0.1592874 0.6770300
```

```
> nextElem(itru)
```

```
 [1] 0.6509264 0.1619922 0.1300430 0.7514514 0.1507951 0.9149560 0.8849137
 [8] 0.2439561 0.6649615 0.2007026
```

The `icount` function returns an iterator that counts starting from one:

```
> it <- icount(3)
> nextElem(it)
```

```
[1] 1
```

```
> nextElem(it)
```

```
[1] 2
```

```
> nextElem(it)
```

```
[1] 3
```