Using The `iterators` Package

Rich Calaway
doc@revolutionanalytics.com

April 10, 2014

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] 0 7 8 1
```

```
> nextElem(ifun)
```

```
[1] 3 1 1 5
```

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

```
> library(foreach)
```

```
foreach: simple, scalable parallel programming from Revolution Analytics
Use Revolution R for scalability, fault tolerance and more.
http://www.revolutionanalytics.com
```

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

```
  [1] -0.0069652059  0.0161112989  0.0080068074 -0.0120020610  0.0017168149
  [6]  0.0139835943 -0.0078172106 -0.0024762273 -0.0031558268 -0.0072662893
 [11] -0.0055142639  0.0015717907 -0.0100842965 -0.0123601527  0.0136420084
 [16] -0.0242922105 -0.0126416949 -0.0052951152  0.0216329326 -0.0262476648
 [21]  0.0041937609  0.0121253368 -0.0110165729  0.0044267635  0.0080241894
 [26]  0.0042995539 -0.0102826632  0.0051185628 -0.0013970812 -0.0172380786
```

```
 [31]  0.0096079613  0.0046837729 -0.0080726970  0.0083781727 -0.0234620163
 [36] -0.0099883966  0.0026883628  0.0029367320  0.0205825899  0.0035303940
 [41]  0.0204990426 -0.0010804987 -0.0033665481 -0.0127492019 -0.0147443195
 [46]  0.0027046346  0.0016449793  0.0155575490 -0.0003488394 -0.0079238019
 [51]  0.0086390030 -0.0039033309  0.0168593825 -0.0067189572 -0.0009925288
 [56] -0.0162907048 -0.0059171838  0.0093806072  0.0100886929 -0.0111677408
 [61]  0.0021754963 -0.0056770907  0.0081200698 -0.0029828717 -0.0163704350
 [66]  0.0057266267 -0.0017156156  0.0214172738 -0.0141379874 -0.0126593342
 [71]  0.0087124575  0.0040231519  0.0038515673  0.0066066908  0.0023586046
 [76] -0.0044167901 -0.0090543553  0.0010806096  0.0102288061  0.0039881702
 [81] -0.0054549319 -0.0127997275 -0.0031697122 -0.0016100996 -0.0143468118
 [86]  0.0035904125 -0.0059399479  0.0085565480 -0.0159064868  0.0054120554
 [91] -0.0084420572  0.0194448129 -0.0103192553 -0.0062924628  0.0215069258
 [96]  0.0015749065  0.0109657488  0.0152237842 -0.0057181022  0.0035530715
```

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]  1.82451011 -0.20727074  0.64879028 -1.65725438  0.62211899  0.42310008
 [7]  0.02620584  1.07414396 -1.13860529  0.63056268

> nextElem(itrn)

 [1]  1.08125364 -1.27105031  0.75554748 -0.85538462  0.30262411 -0.06230941
 [7] -0.70367002  0.76794457  1.08579027  1.79804944
```

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] 0.2707575 0.5250360 0.8132146 0.6020829 0.1674677 0.1540548 0.7942360
 [8] 0.8284512 0.7280557 0.5928254
```

```
> nextElem(itru)
```

```
 [1] 0.96816095 0.18756051 0.32546626 0.65916276 0.30703282 0.04717519
 [7] 0.55762240 0.77949849 0.42856376 0.38335022
```

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
```