Using the Pareto Package

Jacob Simmering

April 13, 2013

1 Introduction

The pareto package includes the standard distribution functions for the Pareto distribution: dpareto() to calculate the density, ppareto() to the cdf, qpareto() to calculate the inverse cdf and rparteo() and rcpareto() provide random samples. The only difference between rpareto() and rcpareto() is that the latter moves most of the $O(N)$ operations to C for speed.

2 Simple Examples

Some simple examples:

```
> dpareto(2, 1, 1)

[1] 0.25

> ppareto(2, 1, 1)

[1] 0.5

> qpareto(0.5, 1, 1)

[1] 2

> rpareto(1, 1, 1)

[1] 1.161902

> rcpareto(1, 1, 1)

[1] 5.185651
```

The functions also support returning or accepting log probabilities as appropriate:

```
> dpareto(2, 1, 1, TRUE)

[1] -1.386294

> ppareto(2, 1, 1, TRUE)

[1] 0.5

> qpareto(0.5, 1, 1, TRUE)

[1] 2
```

3 Vectorized Examples

The functions are also designed to accept vector arguments with lengths greater than 1 element. When provided with this input, the functions will recycle the shorter vectors until they match the longest vector in the input. This is the same as the behavior of other density functions, such as those of the gamma family.

```
> dpareto(1:3, 1:2, 1:2)

[1] 0.0000000 0.0000000 0.1111111

> ppareto(1:4, 1, 1)

[1] 0.0000000 0.5000000 0.6666667 0.7500000

> qpareto(seq(0.1, 0.9, 0.2), 1, 1)

[1]  1.111111  1.428571  2.000000  3.333333 10.000000

> rpareto(5, 1:2, 1:3)

[1] 4.544524 3.269388 1.024007 3.770030 4.079894

> rcpareto(5, 1:2, 1:3)

[1] 7.519426 2.521599 1.412866 2.132816 1.132547
```

The density and cdf of the Pareto(1, 1) distribution is shown in Table 1. A graph of these results is in Figure 1.

	x	dpareto(x)	ppareto(x)
1	1.00	0.00	0.00
2	2.00	0.25	0.50
3	3.00	0.11	0.67
4	4.00	0.06	0.75
5	5.00	0.04	0.80
6	6.00	0.03	0.83
7	7.00	0.02	0.86
8	8.00	0.02	0.88
9	9.00	0.01	0.89
10	10.00	0.01	0.90

Table 1: The density and cdf of the Pareto(1, 1) distribution

4 Parallelization

The functions `p.dpareto, p.ppareto, p.qpareto` have Open MP support for parallel computation. For reasonable lengths of arguments, (e.g., under 10^5 or 10^6) there is no significant increase in performance with parallelization. In fact, for very short inputs (lengths of 100 to 1000), the overhead required by Open MP increases the total computational time relative to the standard functions.

However, for very long inputs, significant speed ups are possible. See Figure **??** and Figure **??** for an illustration of the potential gain by shifting to 2 or 4 cores from the standard function. For instance, with a vector of length $100,000,000$, `dpareto` takes 25.83 seconds. Switching to `p.dpareto` and using 2 cores cuts the time to 15.81 seconds and using 4 cores can cut the time down to 10.96 seconds. This is 38.79% and a 57.57% reduction in run time by using multiple threads.

The functions are used in the same way as the single thread functions, however, they take one additional argument, `P`, which is the number of threads to use in the computation. For example,

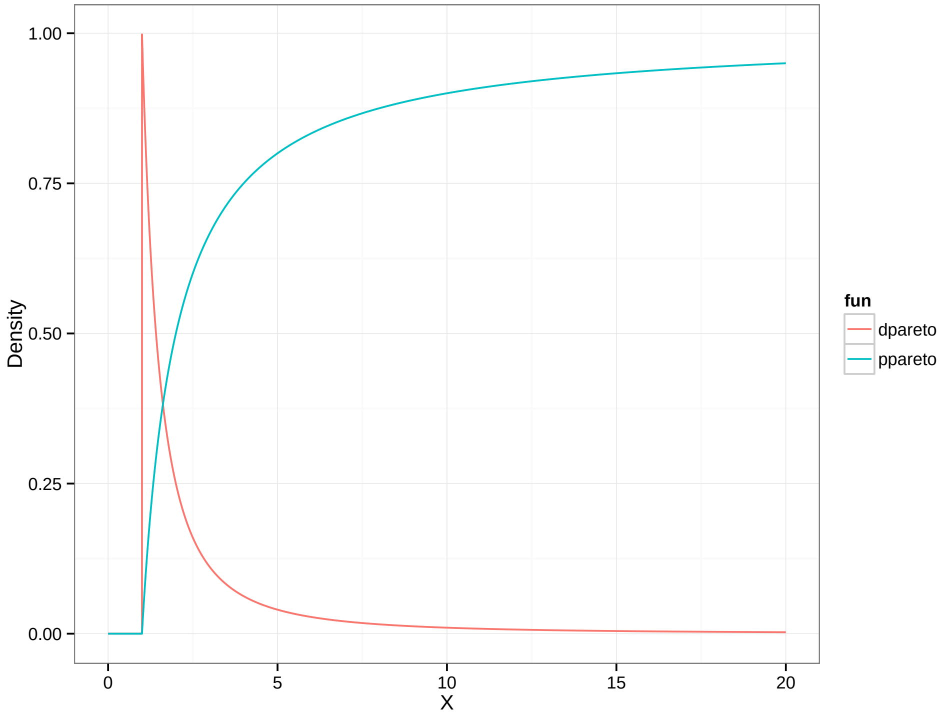

Figure 1: Plots of the Pareto(1, 1) density and cumulative density functions.

```
> p.dpareto(1:3, 1:2, 1:2, P = 2)

[1] 0.0000000 0.0000000 0.1111111

> p.ppareto(1:4, 1, 1, P = 2)

[1] 0.0000000 0.5000000 0.6666667 0.7500000

> p.qpareto(seq(0.1, 0.9, 0.2), 1, 1, P = 2)

[1]  1.111111  1.428571  2.000000  3.333333 10.000000
```