Additive Integer Partitions in R

Robin K. S. Hankin
University of Cambridge

Abstract

This vignette is based on **?**.

This vignette introduces the **partitions** package of R routines, for numerical calculation of integer partititions. Functionality for unrestricted partitions, unequal partitions, and restricted partitions is provided in a small package that accompanies this note; the emphasis is on terse, efficient C code. A simple combinatorial problem is solved using the package.

Keywords: Integer partitions, restricted partitions, unequal partitions, R.

1. Introduction

A *partition* of a positive integer n is a non-increasing sequence of positive integers $\lambda_1, \lambda_2, \ldots, \lambda_r$ such that $\sum_{i=1}^{r} \lambda_i = n$. The partition $(\lambda_1, \ldots, \lambda_r)$ is denoted by λ, and we write $\lambda \vdash n$ to signify that λ is a partition of n. If, for $1 \leqslant j \leqslant n$, exactly f_j elements of λ are equal to j, we write $\lambda = \left(1^{f_1}, 2^{f_2}, \ldots, n^{f_n}\right)$; this notation emphasises the number of times a particular integer occurs as a part. The standard reference is **?**.

The partition function $p(n)$ is the number of distinct partitions of n. Thus, because

$$5 = 4+1 = 3+2 = 3+1+1 = 2+2+1 = 2+1+1+1 = 1+1+1+1+1$$

is a complete enumeration of the partitions of 5, $p(5) = 7$ (recall that order is unimportant: a partition is defined to be a non-increasing sequence).

Various restrictions on the nature of a partition are often considered. One is to require that the λ_i are distinct; the number of such partitions is denoted $q(n)$. Because

$$5 = 4+1 = 3+2$$

is the complete subset of partitions of 5 with no repetitions, $q(5) = 3$.

One may also require that n be split into *exactly* m parts. The number of partitions so restricted may be denoted $r(m, n)$.

2. Package partitions in use

The R (**?**) package **partitions** associated with this paper may be used to evaluate the above functions numerically, and to enumerate the partitions they count. In the package, the number of partitions is given by `P()`, and the number of unequal partitions by `Q()`. For example,

```
> P(100)

[1] 190569292
```

agreeing with the value given by **?**. The unequal partitions of an integer are enumerated by function `diffparts()`:

```
> diffparts(10)

[1,] 10 9 8 7 7 6 6 5 5 4
[2,]  0 1 2 3 2 4 3 4 3 3
[3,]  0 0 0 0 1 0 1 1 2 2
[4,]  0 0 0 0 0 0 0 0 0 1
```

where the columns are the partitions. Finally, function `restrictedpartitions()` enumerates the partitions of an integer into a specified number of parts.

2.1. A combinatorial example

Consider random sampling, with replacement, from an alphabet of a letters. How many draws are required to give a 95% probability of choosing each letter at least once? I show below how the **partitions** package may be used to answer this question exactly.

A little thought shows that the number of ways to draw each letter at least once in n draws is

$$N = \sum_{\lambda \vdash n;a} \frac{a!}{\prod_{i=1}^{r} f_i!} \cdot \frac{n!}{\prod_{j=1}^{n} \lambda_i!} \qquad (1)$$

where the sum extends over partitions λ of n into exactly a parts; the first term gives the number of ways of assigning a partition to letters; the second gives the number of distinct arrangements.

The corresponding R idiom is to define a nonce function `f()` that returns the product of the two denominators, and to sum the requisite parts by applying `f()` over the appropriate restricted partitions. The probability of getting all a letters in n draws is thus N/a^n, computed by function `prob()`:

```
> f <- function(x) {
+     prod(factorial(x), factorial(tabulate(x)))
+ }
> prob <- function(a, n) {
+     jj <- restrictedparts(n, a, include.zero = FALSE)
+     N <- factorial(a) * factorial(n) * sum(1/apply(jj, 2, f))
+     return(N/a^n)
+ }
```

In the case of $a = 4$, we obtain $n = 16$ because `prob(4,15)` $\simeq 0.947$ and `prob(4,16)` $\simeq 0.96$.

3. Conclusions

The **partitions** package was developed to answer the combinatorial word question discussed above: it does so using fast C code. Further work would include the enumeration of compositions and vector compositions.

Acknowledgement

I would like to acknowledge the many stimulating and helpful comments made by the R-help list while preparing the package.

Affiliation:

Robin K. S. Hankin
The University of Cambridge
19 Silver Street Cambridge CB3 9EP United Kingdom
E-mail: rksh1@cam.ac.uk