

Enumerations

- In certain calculations it is necessary to list all the elements in a big set. Examples include:
 1. The $n!$ permutations of n objects.
 2. The b^n maps from n objects into b values.
 3. The set of partitions $Z = \sum_{i=1}^n A_i$ with $A_i \geq 0$.
 4. The contingency tables with a given set of row and column marginals.
- The basic techniques are either (i) recursion or (ii) assigning an integer key to each object in the set and cycling through the keys.
- The sets described above grow exponentially in n . The enumeration algorithms described here are therefore useful for small or moderately small values of n . For larger values of n Monte Carlo methods can be used.

Maps

- A *map* is a function from a domain n discrete objects to a range of b discrete objects. There are b^n distinct such maps.

A map ϕ can be viewed as a way of placing n objects into b categories, where $\phi(k)$ is the category to which object k belongs.

If $b = 2$ then the maps from \mathcal{S} into $\{0, 1\}$ are in one-to-one correspondence with the set of subsets of \mathcal{S} . The map $\phi : \mathcal{S} \rightarrow \{0, 1\}$ defines a subset $\mathcal{U}_\phi \subset \mathcal{S}$ by $\mathcal{U}_\phi = \{u \in \mathcal{S} | \phi(u) = 1\}$.

- A *key* is a one-to-one association between an arbitrary finite set and the integers $0, 1, \dots, m - 1$. Once a key is available, the set can be enumerated by cycling through the integers from 0 to $m - 1$ and constructing each object from its key.
- A key for the set of all maps of n objects into b groups can be obtained from the base b expansion of an integer between 0 and $b^n - 1$:

$$d = \sum_{k=0}^{n-1} a_k b^k,$$

where $a_k = \lfloor d/b^k \rfloor \pmod{b}$. This expansion is a way of associating a unique integer to each sequence (a_0, \dots, a_{n-1}) such that $0 \leq a_i < b$ for each i .

For example, suppose we wish to enumerate all mappings of $n = 4$ objects into $b = 2$ classes. The key/map pairs are as follows:

$(0,0,0,0)$	\rightarrow	0	$(1,0,0,0)$	\rightarrow	8
$(0,0,0,1)$	\rightarrow	1	$(1,0,0,1)$	\rightarrow	9
$(0,0,1,0)$	\rightarrow	2	$(1,0,1,0)$	\rightarrow	10
$(0,0,1,1)$	\rightarrow	3	$(1,0,1,1)$	\rightarrow	11
$(0,1,0,0)$	\rightarrow	4	$(1,1,0,0)$	\rightarrow	12
$(0,1,0,1)$	\rightarrow	5	$(1,1,0,1)$	\rightarrow	13
$(0,1,1,0)$	\rightarrow	6	$(1,1,1,0)$	\rightarrow	14
$(0,1,1,1)$	\rightarrow	7	$(1,1,1,1)$	\rightarrow	15

- To prove uniqueness, suppose $\sum a_k b^k = \sum a'_k b^k$, and let r be the least integer such that $a_r \neq a'_r$. Then we can write

$$a_r - a'_r = \sum_{k=r+1}^{n-1} (a'_k - a_k) b^{k-r}.$$

The right side is divisible by b , but the left side is strictly smaller than b in magnitude, so no such r exists.

- The smallest key is clearly zero, and the largest key is

$$\sum_{k=0}^{n-1} (b-1)b^k = \frac{b-1-b^{n+1}+b^n}{1-b}$$

which is easily seen to be equal to $b^n - 1$. Thus there are n keys, and since the association of a map to its key is one-to-one, and there are the same number of maps as keys, the association of a map to its key must also be onto (pigeonhole principle).

- **Example:** Suppose we have a $n \times m$ matrix with each of the n rows representing an observed object, and the m columns being features describing the objects. We may wish to partition the n objects into two groups such that a clustering score equal to the total of all absolute differences between the feature means in the two groups is maximized.

If n is small, we can do this by exhaustively enumerating all splits into two groups and selecting the one with the optimal clustering score.

Octave code for exhaustively optimizing the cluster score:

```
## Number of variables.
m = 20;

## Number of objects.
n = 12;

## The map: I(k) = 1 or 0 denoting the group to which object k belongs.
I = zeros(n,1);

## The current best score.
B = -1;

while (1)

    ## Need at least one object in each group.
    if ( (sum(I) < n) && (sum(I) > 0) )

        MD = sum(abs(mean(Z(find(I==1),:)) - mean(Z(find(I==0),:))));

        if (MD > B)
            IB = I;
            B = MD;
        endif

    endif

    ## Move to the next map.
    for k=1:n
        if (I(k) == 1)
            I(k) = 0;
        else
            I(k) = I(k)+1;
            break;
        endif
    endfor

    ## Stop once we've seen every map distinct split (we don't need to
    ## consider both I and 1-I since they give the same split).
    if (I(n) == 1)
        break;
    endif

endwhile
```

Permutations

- A *permutation* is a one-to-one mapping of a set \mathcal{S} onto itself. If the cardinality of \mathcal{S} is m there are $m!$ distinct permutations.

The standard representation of a permutation on m objects is a vector $\pi_j \in \{0, \dots, m-1\}^m$ in which each value occurs exactly once. These objects act as functions, for example:

$$(3, 1, 4, 2) \circ (A, B, C, D) = (C, A, D, B).$$

- A key for the set of permutations on m objects can be constructed by generalizing the base b expansion of an integer. We need the following two results.

Lemma 1: Let $0 < d_1 < d_2 < \dots < d_m$ be integers, and let $D_k = \prod_{j=1}^k d_j$. Then for any sequence of integers U_1, \dots, U_m with $0 \leq U_j \leq d_j$, the association between the sequence (U_1, \dots, U_m) and the integer $\sum_{j=1}^m U_j D_j$ is invertible. That is, if U_1^*, \dots, U_m^* is another sequence of integers satisfying $0 \leq U_j^* \leq d_j$, then $\sum_j U_j D_j = \sum_j U_j^* D_j$ implies $U_j = U_j^*$ for each j .

Proof of lemma 1: Suppose $U_j = U_j^*$ for $j < \ell$, so

$$\sum_{j=\ell}^m U_j D_j = \sum_{j=\ell}^m U_j^* D_j,$$

and moreover

$$\sum_{j=\ell}^m U_j \tilde{D}_j = \sum_{j=\ell}^m U_j^* \tilde{D}_j,$$

where $\tilde{D}_j = D_j/D_\ell = \prod_{k=\ell+1}^j d_k$ are integers. Setting $A_j = U_j - U_j^*$, we have

$$\sum_{j=\ell}^m A_j \tilde{D}_j = 0.$$

Since $\tilde{D}_\ell = 1$,

$$-A_\ell = \sum_{j=\ell+1}^m \tilde{D}_j A_j.$$

Since $d_{\ell+1}$ divides each term on the right, it must divide A_ℓ , but $|A_\ell| \leq d_\ell < d_{\ell+1}$, so $A_\ell = 0$.

Lemma 2: The map from (U_1, \dots, U_m) to $\sum_{j=1}^m U_j D_j$ is onto if and only if $d_1 = 1, d_2 = 2, \dots$ in which case the range is the integers between 0 and $m! - 1$.

Proof of lemma 2: We only require the forward implication. First note that by induction

$$\sum_{j=1}^{m-1} j \cdot j! = m! - 1,$$

and since $D_k = k!$, this is the largest possible key. Since (i) we already know that the mapping is one-to-one and (ii) there are $m!$ distinct input sequences, then if the largest value is equal to $m! - 1$ then every value between 0 and $m! - 1$ (inclusive) must be in the image by the pigeonhole principle.

- **Keys for permutations:** We will use the generalized expansion whose properties are given in the preceding lemmas as keys for permutations. To do this we need to adopt a different representation of a permutation.

Let $\tilde{\pi}_1 = \pi_1$, and let $\tilde{\pi}_j$ be equal to the position of π_j in the ordered list

$$\{0, 1, \dots, m-1\} \setminus \{\pi_1, \dots, \pi_{j-1}\}.$$

Note that $0 \leq \tilde{\pi}_j \leq m-j$ and $\tilde{\pi}_m = 0$. This leads immediately to a one-to-one association of each permutation to an integer between 0 and $m! - 1$. Some examples of the relationship between π , $\tilde{\pi}$, and the key when $m = 4$ follow:

$\pi = (2, 0, 3, 1)$	$\tilde{\pi} = (2, 0, 1)$	13
$\pi = (1, 3, 0, 2)$	$\tilde{\pi} = (1, 2, 0)$	10
$\pi = (0, 3, 2, 1)$	$\tilde{\pi} = (0, 2, 1)$	5
$\pi = (3, 0, 2, 1)$	$\tilde{\pi} = (3, 0, 1)$	19.

R code for listing permutations using keys:

```
key_to_perm <- function(K, n)
{
  P <- array(0, n)

  ## Get the compressed representation.
  P[1] <- 0
  for (i in (2:n))
  {
    K <- floor(K/(i-1))
    P[i] <- K %% i
    K <- K - P[i]
  }

  ## Convert to the standard representation.
  for (j in (1:n)) { P[j+n] <- j-1 }
  for (j in seq(n,1))
  {
    d <- P[n+1+P[j]]
    for (i in (P[j]:(n-1))) P[n+i+1] <- P[n+i+2]
    P[j] <- d
  }

  return(P[1:n])
}

p <- 3

for (k in (0:(factorial(p)-1)))
{
  print(key_to_perm(k, p))
}
```

- **Permutations via recursion:** A completely different approach is to generate permutations using recursion. Suppose we can generate all permutations on the set $\{1, \dots, n-1\}$. Then all permutations on $\{1, \dots, n\}$ can be obtained by taking each permutation on the smaller set and inserting n at each possible point. For example, if $n = 3$ then the permutation $(1, 3, 2)$ would be extended to $(1, 3, 2, 4)$, $(1, 3, 4, 2)$, $(1, 4, 3, 2)$, $(4, 1, 3, 2)$.

R code for listing permutations using recursion:

```
listperm <- function(n)
{
  ## Bail out of the recursion.
  if (n == 1) { return(list(c(1))) }

  ## Storage for the result and a position variable.
  Q <- list()
  q <- 1

  ## Recur.
  P <- listperm(n-1)

  ## Extend all the permutations in P.
  for (p in P)
  {
    for (j in (1:n))
    {
      ii <- 1
      w <- array(0, n)
      w[j] <- n
      for (i in (1:n))
      {
        if (i != j)
        {
          w[i] <- p[ii]
          ii <- ii+1
        }
      }
      Q[q] <- list(w)
      q <- q+1
    }
  }

  return(Q)
}
```

Partitions

- A *partition* is an additive decomposition $N = \sum_{i=1}^k A_i$, where $0 \leq A_i \leq M_i$, and N and the M_i are given. For small values of k this can be achieved by constructing a series

of nested loops. The following Octave code fragment demonstrates this approach for a problem with $k = 4$:

Octave code for listing partitions via explicit loops:

```

for a=0:min(N, M(1))
  for b=0:min(N-a, M(2))
    for c=0:min(N-a-b, M(3))
      d = N-a-b-c;
      if (d > M(4))
        continue;
      endif

      ## Do something with the partition here.

    endfor
  endfor
endfor

```

For example, here is the output of this code using $N = 5$, $M = (5, 5, 5, 3)$:

```

0 0 2 3      1 0 2 2      2 0 0 3      3 0 0 2      4 0 0 1      5 0 0 0
0 0 3 2      1 0 3 1      2 0 1 2      3 0 1 1      4 0 1 0
0 0 4 1      1 0 4 0      2 0 2 1      3 0 2 0      4 1 0 0
0 0 5 0      1 1 0 3      2 0 3 0      3 1 0 1
0 1 1 3      1 1 1 2      2 1 0 2      3 1 1 0
0 1 2 2      1 1 2 1      2 1 1 1      3 2 0 0
0 1 3 1      1 1 3 0      2 1 2 0
0 1 4 0      1 2 0 2      2 2 0 1
0 2 0 3      1 2 1 1      2 2 1 0
0 2 1 2      1 2 2 0      2 3 0 0
0 2 2 1      1 3 0 1
0 2 3 0      1 3 1 0
0 3 0 2      1 4 0 0
0 3 1 1      1 0 1 3
0 3 2 0
0 4 0 1
0 4 1 0
0 5 0 0

```

- **Partitions via loop emulation:** When k is greater than four or so, it is cumbersome to program the loops explicitly. The same results can be achieved by emulating the behavior of k nested loops in a single loop. The basic idea is to store all the “loop

index variables” in an array $I = [i_1, \dots, i_{k-1}]$, starting with $I = [0, \dots, 0]$, and at each iteration perform the following steps:

1. Let $D = N - \sum_{\ell} i_{\ell}$. If $D \leq M_k$ add $[i_1, \dots, i_{k-1}, D]$ to the list of partitions.
2. Set $j = k - 1$.
3. If $i_j + 1 \leq M_j$ set $i_j = i_j + 1$, go to step 1.
4. Set $i_j = 0$, set $j = j - 1$. If $j < 1$, stop. Otherwise repeat step 3.

Octave code for listing partitions via loop emulation:

```
## The sum.
N = 5;

## Upper bounds for the partition elements.
M = [5, 5, 5, 3];

## The vector of loop index variables (the last index is determined by the
## others so is not included).
I = [0, 0, 0];

while (1)

    ## Attempt to extend I to a partition, if successful,
    ## do something with the result.
    ii = N - sum(I);
    if ( (ii >= 0) && (ii <= M(4)) )
        disp([I, ii]);
    endif

    ## Increment to the next partition.
    for k=1:length(I)
        if (I(k) < M(k))
            I(k) = I(k)+1;
            break;
        else
            I(k) = 0;
        endif
    endfor

    ## Test for completion.
    if (all(I==0))
        break;
    endif

endwhile
```

- **Partitions via recursion:** Another natural strategy for enumerating the partitions is to use recursion. For each $0 \leq a \leq M_1$ find the set Q_a of partitions of $N - a$ in $k - 1$ summands with bounds M_2, \dots, M_k . The solution to the original problem is the set of all $\{(a, b) | a = 0, 1, \dots, M_1, b \in Q_a\}$.

Contingency tables

- Suppose we want to enumerate all $m \times n$ contingency tables with row marginals given by R and column marginals given by C . That is, we want to consider all $m \times n$ arrays T of non-negative integers that satisfy:

$$\begin{aligned}\sum_j T_{ij} &= R_i, \quad i = 1, \dots, m \\ \sum_i T_{ij} &= C_j, \quad j = 1, \dots, n.\end{aligned}$$

- This enumeration has applications in hypothesis testing. Suppose we have a test statistic G and we observe $G_{\text{obs}} = G(T)$. Then if our null hypothesis is that every table is equally likely, the p-value is

$$p = P(G(T) > G_{\text{obs}}).$$

On the other hand it is often more powerful to use the conditional p-value

$$p_c = P(G(T) > G_{\text{obs}} | R, C).$$

This strategy is employed, for example, in Fisher's exact test. See the review paper by Agresti (Statistical Science, Vol. 7, No. 1, Feb., 1992, pp. 131-153) for details.

- The tables can be enumerated recursively. We make use of the partition enumeration described in the previous section.
 1. If $m = 1$ there are either 0 or 1 solutions, depending on whether $R_1 = \sum_j C_j$.
 2. If $m > 1$, compute all partitions $R_1 = \sum_{j=1}^n G_j$ with $0 \leq G_j \leq C_j$. For each of these partitions construct adjusted column marginals $C_j^* = C_j - G_j$. By recursion we can compute all $m - 1 \times n$ tables with row marginals R_2, \dots, R_m and column marginals C_1^*, \dots, C_n^* . Appending C_1^*, \dots, C_n^* as the first row of these tables produces all $m \times n$ tables with row marginals R_j and column marginals C_j .

For example, here are the 3×2 contingency tables with row sums $(2, 3, 2)$ and column sums $(3, 4)$.

0 2	1 1	2 0
1 2	0 3	0 3
2 0	2 0	1 1

0 2	1 1	2 0
2 1	1 2	1 2
1 1	1 1	0 2

0 2	1 1
3 0	2 1
0 2	0 2