Using R to Compute Probabilities of Matching Birthdays

by Bruce E. Trumbo, Eric A. Suess, and Clayton W. Schupp

In this article we introduce the statistical package R and use it to model and generalize a famous problem about matching birthdays. The R language is being used more and more widely in applied probability modeling and research. Also, R is easy enough to learn that beginning students can use it for basic statistics and probability computations. Maybe best of all, R is available free of charge on the web [1]. Therefore, R is a natural choice for use in beginning statistics courses [2].

Statement of the Birthday Problem. Suppose that there are 25 randomly chosen people in a room. What is the probability that two or more of them have the same birthday? Ignore leap years, and assume that the 365 days of the year are equally likely birthdays. You may have seen this problem before. Since Feller introduced it in his widely-used introductory probability text in 1950, this problem has become a classic [3].

Solution. We find the probability of *no matches* by considering the 25 people one at a time. Obviously, the *first* person chosen cannot produce a match. The probability that the *second* person is born on a different day of the year than the first is 364/365 = 1 – 1/365. The probability that the *third* person avoids the birthdays of the first two is 363/365 = 1 – 2/365, and so on to the *25th* person. Thus we have

$$\text{P(No Match)} = \frac{P_{25}^{365}}{365^{25}} = \prod_{i=0}^{24}\left(1-\frac{i}{365}\right) = 0.4313,$$

and P(At Least 1 Match) = 1 – 0.4313 = 0.5687. The computation is a bit tedious because it involves multiplying 25 factors. We use R to get the answer.

R is based on vectors. (If you don't know about vectors, it is good enough for this article to think of them as ordered lists of numbers, called elements.) For example, in R the vector (0, 1, 2, ..., 24) can be written as `0:24`. Any arithmetic involving a vector and a single number is done element by element. (We show a few simple examples in the next section.) Thus the vector consisting of the 25 factors in the big parentheses in the displayed equation above can be written `1 - (0:24)/365`. The R function `prod` takes the product of the elements of a vector. Display 1 shows how the computation of P(No Matches) looks in the "R-Console" window.

Display 1

```
> prod(1 - (0:24)/365)
[1] 0.4313003
```

For a room with as few as 25 people, some beginning students are surprised that the probability of duplicate birthdays is so large—above 1/2. Perhaps they are thinking about a different problem: "If I were in the room, it seems that the chances of someone else having *my* birthday would be very small." The question is not to find the probability of duplicating any one person's birthday, but the chances that *any pair* of people in the room have birthdays that match.

Some Basic Ideas of R. Before we explore the birthday problem further, we illustrate a few simple facts about how R works. (After the `>` prompt in the R-Console window, you can type the expressions we show in `typewriter type`.)

Defining a vector. Because the R language is based on vectors, we begin by showing a few ways to specify vectors. The operator `<-` is used to assign values. For example, `n <- 25` means that `n` is a vector with 25 as its single element. The "combining" symbol `c` can be used to make vectors with several elements:

`v1 <- c(7, 2, 3, 5)` means $v_1 = (7, 2, 3, 5)$.

Here are a few convenient shorthand notations for making vectors, along with their meanings expressed in standard mathematical notation:

`v2 <- numeric(4)` means $v_2 = (0, 0, 0, 0)$,
`v3 <- rep(3, 4)` means $v_3 = (3, 3, 3, 3)$,
`v4 <- 1:4` means $v_4 = (1, 2, 3, 4)$, and
`v5 <- c(v3, v4, 7)` is the combination of three vectors, giving $v_5 = (3, 3, 3, 3, 1, 2, 3, 4, 7)$.

Simple arithmetic. Operations are elementwise.

`w1 <- 3*v1` means $w_1 = (21, 6, 9, 15)$,
`w2 <- v1/2` means $w_2 = (3.5, 1, 1.5, 2.5)$,
`w3 <- 5 - v1` means $w_3 = (–2, 3, 2, 0)$,
`w4 <- w3^2` means $w_4 = (4, 9, 4, 0)$, and
`w5 <- w3 + w4` means $w_5 = (2, 12, 6, 0)$.

Indexes and assignments. Sometimes we want to deal with only one element of a vector. The index notation `[]` helps to do this. The simplest use of indexing is just to specify the index (position number) you want.

`w1[3]` returns 9, `v5[9]` returns 7, and
`v2[1] <- 6` changes v_2 so that $v_2 = (6, 0, 0, 0)$.

When there are *n* people in the room. As the number n of people in the room increases, it is clear that the probability of matching birthdays increases. Of course, if $n = 366$ (still ignoring leap years), we are *sure* to get at least one duplication. But we will see that the probability of at least one match becomes very close to 1 for much smaller values of n.

With R it is not difficult to let n run through a suitable number of values (finding the probability of

matches for each value of n) and then to make a plot of the results. (See Figure 1.) The R script in Display 2 shows how to do this. We chose to let n loop through the values from 1 to 50, where we determined by trial and error that 50 is about big enough.

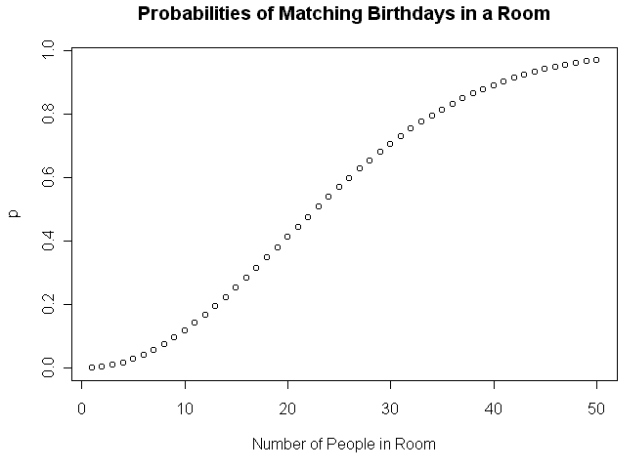
Display 2

```
p <- numeric(50)
for (n in 1:50)
   {
   q <- 1 - (0:(n - 1))/365
   p[n] <- 1 - prod(q)
   }
plot(p)         # Makes Figure 1
p               # Makes prinout below

> p
 [1] 0.000000000 0.002739726
  ...
[21] 0.443688335 0.475695308
[23] 0.507297234 0.538344258
[25] 0.568699704 0.598240820
  ...
[49] 0.965779609 0.970373580
```

When the loop is completed, 50 values have been put into the vector `p`. For each value of $n = 1, ..., 50$, the corresponding element is the probability of finding at least one repeated birthday in a room with n people. To save space, the printout of the vector `p` has been abridged in Display 2. The numbers in brackets give the index (value of n) of the *first* result printed on each line. In particular, the 25th element of `p` is 0.5687. The 23rd element is the first to exceed 1/2. The printout in Display 2 and the plot in Figure 1 both show that in a room with as many as 50 people we are very likely to see some matching birthdays.

Figure 1: The probability P(At Least 1 Match) increases from 0 to near 1 as the number of people in the room increases from 1 to 50.



More about R. Before we return to the birthday problem for a deeper look, we pause to say a little more about R.

Some vector functions. Notice that most of the functions we show below return single numbers. But `unique` returns a vector; it is a crucial function later in this article.

`max(w2)` returns 3.5, `mean(w3)` returns 0.75, `sum(v1)` returns 17, `prod(v4)` returns 24, and `length(v5)` returns 9.

`x2 <- unique(v5)` means $x_2 = (3, 1, 2, 4, 7)$, as "redundant" elements are eliminated, so that `length(unique(v5))` returns 5.

Comparisons and logical values. So far, we have considered only vectors of numbers, but R can also use vectors of "logical" values: `T` for True and `F` for False. Sometimes these values arise from comparisons. In this article we use the comparison operator `==`, which checks to see if numerical values are equal. If R is "forced" to do arithmetic on logical values, then `T` is taken to be 1, and `F` to be 0.

`y <- (w4==4)` means $y = (T, F, T, F)$, and `mean(y)` returns 0.5: half of the values of `w4` are 4s.

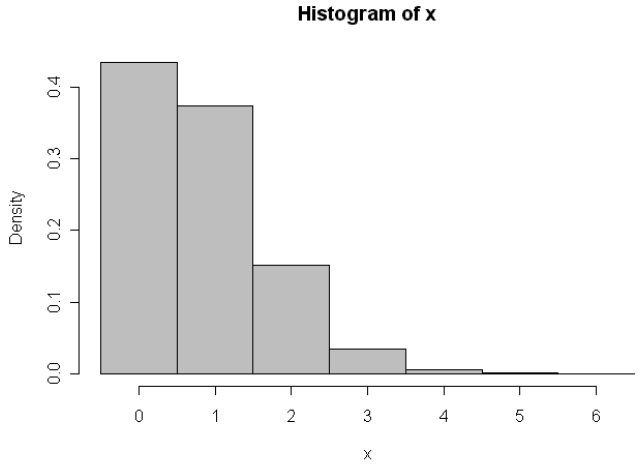
Sampling from a finite population. The `sample` function selects a sample of a specified size from a given population. For example, `sample(1:365, 1)` takes one person's birthday at random from among the numbers 1, 2, ..., 365. The first argument is the population and the second is the sample size. We used this function three times, obtaining random birthdays 106, 182, and 140.

If the sample size is two or more, we have to specify whether sampling is to be done with replacement. To sample birthdays of 25 randomly chosen people, we could use `sample(1:365, 25, repl=T)`, where the third argument indicates that sampling is with replacement. Similarly, `sample(1:6, 2, repl=T)` simulates rolling a pair of dice, and `sample(1:52, 13)` simulates a bridge hand, where sampling is *without* replacement (the default `repl=F` need not be specified).

Of course, there is a lot more to R than we have shown here, but we have shown enough for now. (The best way to learn R, or any other multi-purpose software package, is just to plunge in, learning parts of it as needed.)

Simulating the Birthday Process. Next, we use R to simulate the process of looking for matching birthdays among $n = 25$ people in a room. This is an entirely different approach from the probability computations we did earlier. Now we will simulate the birthday process many times and summarize the results.

Figure 2: The simulated distribution of the number of birthday matches in a room with 25 randomly chosen people. The height of the left hand bar estimates P(No Matches).



This approach allows us to find the approximate distribution of the number X of duplicate birthdays. From this simulated distribution, we can approximate $P(X = 0)$, which we already know to be 0.4313, and we can also approximate $E(X)$, which would be somewhat difficult to find without simulation methods. Later in this article we will show how simulation methods allow us to investigate the importance of some of the assumptions we have made so far in solving the birthday problem. Now we build the simulation model step by step.

(1) Simulating birthdays for 25 people in a room. We have already seen that this can be done with the sample function. We put the results into a vector `b`, which has 25 elements.

```
b <- sample(1:365, 25, repl=T).
```

(2) Finding the number of birthday matches among 25 people. We use the `unique` function to find the number of different birthdays, then subtract from 25 to find the number X of birthday matches ("redundant" birthdays):

```
x <- 25 - length(unique(b)).
```

(3) Using a loop to simulate X for many rooms. Repeat the process for $m = 10{,}000$ rooms. In this simulation `x` is a vector with `m` elements. The population mean $E(X) = \mu_X$ is approximated by `mean(x)`, the sample mean of the m simulated values of X. The population probability $P(X = 0)$ is approximated by `mean(x==0)`, the sample *proportion* of Xs equal to 0. And `hist` makes a histogram (Figure 2) of the simulated distribution of X. (The parameters of `hist` are chosen to make a nice looking graph.) The complete R script is shown in Display 3 along with the results of one of our runs. (Semicolons separate multiple statements on a line of code.)

Display 3

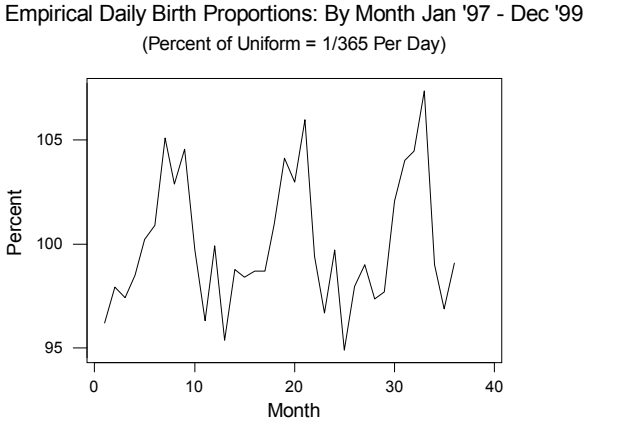
```
n <- 25;  m <- 10000;  x <- numeric(m)
for (i in 1:m)  {
   b <- sample(1:365, n, repl=T)
   x[i] <- n - length(unique(b))  }
mean(x);  mean(x==0)
cut <- (0:(max(x) + 1)) - 0.5
hist(x, breaks=cut, freq=F, col=8)

> mean(x)
[1] 0.8081
> mean(x==0)
[1] 0.4338
```

The simulated value 0.4338 of the probability of seeing no matches agrees well with the known exact value 0.4313. Additional runs of the program consistently gave values of $E(X)$ in the range 0.81 ± 0.02.

Checking Assumptions. In modeling any real life situation we must make assumptions. Here we hope that the people in the room are randomly chosen from the population at large — for example, *not* a group of twins or of people born in Sagittarius. Even though we know it is not true, we assume that there are no leap years and that the birth rate is uniform throughout the year. How sensitive is our computation of the probability of duplicate birthdays to the assumption that there are 365 equally likely birthdays? A slightly more general simulation can answer this question.

Figure 3: Cyclical pattern of US birth frequencies for 36 consecutive months. Daily birth proportions typically exceed 1/365 from May through September.



The `sample` function can take samples from a specified population—with specified probabilities for each element of the population. Now assume that 200 days of the year have below average birthrates and 165 days have above average birthrates. Also, let's account for leap years. Specifically, we model the birthrates on various days of the year according to 366 weights `w` given by

```
w <- c(rep(4,200), rep(5,165), 1).
```

Then the *n* birthdays can be sampled with replacement and with our designated probabilities from among the numbers 1, 2, ..., 366 by using the function

```
b <- sample(1:366, n, repl=T, prob=w).
```

The last parameter gives proportional weights or probabilities for each population element. The population and weighting vectors should be of equal length, here 366. In use, R multiplies the 366 weights by a constant so that they sum to 1. So here the weight 0.002460 = 4/[200(4) + 165(5) + 1] (which is 89.8% of uniform 1/365) appears 200 times; 0.003075 (112.2% of 1/365) appears 165 times; and 0.000615 represents February 29.

Actual 1997-99 data for the US show slightly less variation in daily birth proportions than we used in our simulation [4]. Monthly averages range from a low of about 94.9% of uniform in January 1999 to a high of about 107.4% in September 1999. See Figure 3.

Display 4 shows the minor change in the script of Display 3 that is needed in order to implements the weights. (Equal weights are used unless the `prob` parameter is specified.)

Display 4

```
n <- 25;  m <- 20000;  x <- numeric(m)
w <- c(rep(4,200), rep(5,165), 1)
for (i in 1:m)  {
   b <- sample(1:366, n, repl=T, prob=w)
   x[i] <- n - length(unique(b))  }
mean(x);  mean(x==0)
```

```
> mean(x)
[1] 0.819
> mean(x==0)
[1] 0.42215
```

The values $P(X = 0) \approx 0.42$ and $E(X) \approx 0.82$ based on a particular pattern of birthdays that are not equally likely are not very much different from the results we obtained under the assumption of equally likely birth days. From this and related simulations, we conclude that, although birthdays are not actually uniformly distributed, it seems harmless in practice to assume they are.

When one abandons the assumption that birthdays are equally likely, direct computation of P(No Match) is no longer elementary [5]. But simulation is almost as easy as in the uniform case. There are many real-life situations in which the only feasible way to check assumptions is by means of simulation.

References and Web Resources.

[1] R software and manuals are available at
`www.r-project.org`

[2] Peter Dalgaard: *Introductory Statistics with R,* 2002, Springer.

[3] William Feller: *An Introduction to Probability Theory and Its Applications,* Vol. 1, 1950 (3rd ed., 1968), Wiley.

[4] Birth frequencies are based on raw monthly totals available at
`www.cdc.gov/nchs/products/pubd/vsus/vsus.html`

[5] Thomas. S. Nunnikhoven: A birthday problem solution for nonuniform frequencies, *The American Statistician,* 46, 270–274, 1992.

Auxiliary instructional materials for this article are posted at
`www.sci.csuhayward.edu/~btrumbo/birthmatch`

Authors: Bruce Trumbo and Eric Suess are faculty members in the Statistics Department at California State University, Hayward; Hayward, CA 94542 USA. Email: `btrumbo@csuhayward.edu`..

Clayton Schupp is an MS student in the Department. He also works in the Quality Metrics department at Sun Microsystems, currently analyzing CPU failure rates.