USING R TO COMPUTE PROBABILITIES OF MATCHING BIRTHDAYS

by Bruce E. Trumbo, Eric A. Suess, and Clayton W. Schupp
Department of Statistics; California State University, Hayward; Hayward, CA 94542 USA;
btrumbo@csuhayward.edu or esuess@csuhayward.edu

Abstract: Appropriate simulations can enliven a beginning probability course by focusing on model building, exploring generalizations that would lead to analytically intractable results, and teaching computer skills that are valuable in the job market. Here we use R to simulate the probability of birthday matches in a room with n people. The first simulation model assumes a uniform distribution of birthdays throughout the year, giving results that agree with a standard elementary formula. A very similar second simulation model drops the uniformity assumption, giving results that are analytically beyond the level of undergraduate mathematics. Data for the actual distribution of US birthdays in 1997-99 provide a basis for the second model. An entry-level exposition of the required functions in R is provided.

Key Words: Birthday matches, nonuniform birthday distribution, probability modeling, teaching; R.

1. Introduction.
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 has become popular 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, and R is available free of charge on the web [1]. So R is a natural choice for use in beginning courses [2].

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 [3], this problem has become a classic.

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

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

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

R is based on vectors. (For this article it is enough to view vectors as ordered lists of numbers, called elements.) For example, in R the vector \((0, 1, 2, \ldots, 24)\) 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 we can write the vector of the 25 factors in the big parentheses in the displayed equation above as \(1 - (0:24)/365\). The R function `prod` takes the product of the elements of a vector. In summary, Display 1 shows how the computation of \(P(\text{No Matches})\) looks in the "R-Console" window.

Display 1

\[ \text{> prod}(1 - (0:24)/365) \]
\[ [1] \ 0.4313003 \]

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.

3. 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 \(v1 = (7, 2, 3, 5)\).

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

\[ v2 <- \text{numeric}(4) \]

means \(v2 = (0, 0, 0, 0)\),

\[ v3 <- \text{rep}(3, 4) \]

means \(v3 = (3, 3, 3, 3)\),

\[ v4 <- 1:4 \]

means \(v4 = (1, 2, 3, 4)\), and

\[ v5 <- c(v3, v4, 7) \]

puts three vectors together to give \(v5 = (3, 3, 3, 3, 1, 2, 3, 4, 7)\).

Simple arithmetic. Operations are elementwise.

\[ w1 <- 3*v1 \]

means \(w1 = (21, 6, 9, 15)\),

\[ w2 <- v1/2 \]

means \(w2 = (3.5, 1.5, 1.5, 2.5)\),

\[ w3 <- 5 - v1 \]

means \(w3 = (-2, 3, 2, 0)\),

\[ w4 <- c(w1, w2, w3) \]

puts three vectors together to give \(w4 = (21, 6, 9, 15, 3.5, 1.5, 1.5, 2.5, -2, 3, 2, 0)\).
\[
\begin{align*}
w_4 &= -w_3^2 \text{ means } w_4 = (4, 9, 4, 0), \\
w_5 &= -w_3 + w_4 \text{ means } w_5 = (2, 12, 6, 0).
\end{align*}
\]

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

\[
w_1[3] \text{ returns } 9, \quad v_5[9] \text{ returns } 7, \quad \text{and} \quad v_2[1] \gets 6 \text{ changes } v_2 \text{ so that } v_2 = (6, 0, 0, 0).
\]

### 4. 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 choose to let \(n\) loop through the values from 1 to 50 (after using trial and error to decide that 50 is about big enough).

When the loop is completed, 50 values have been put into the vector \(p\). For each value of \(n = 1, \ldots, 50\), 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.

### 5. More about R.

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

#### Display 2

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

\[
> \text{p}
\]
\[
\begin{align*}
[1] & 0.000000000 0.002739726 \\
[21] & 0.443688335 0.475695308 \\
[23] & 0.507297234 0.538344258 \\
[25] & 0.568699704 0.598240820 \\
[49] & 0.965779609 0.970373580
\end{align*}
\]

**Some vector functions.** Most of the functions we show below return single numbers. But unique is different; it returns a vector. This function is crucial for our computation in Section 6.

\[
\text{max}(w_2) \text{ returns } 3.5, \quad \text{mean}(w_3) \text{ returns } 0.75, \\
\text{sum}(v_1) \text{ returns } 17, \quad \text{prod}(v_4) \text{ returns } 24, \quad \text{and length}(v_5) \text{ returns } 9.
\]

\[
x_2 \gets \text{unique}(v_5) \text{ means } x_2 = (3, 1, 2, 4, 7), \quad \text{as "redundant" elements are eliminated, so that} \\
\text{length(unique}(v_5)) \text{ returns } 5.
\]

**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.

\[
y \gets (w_4==4) \text{ means } y = (T, F, T, F), \quad \text{and} \quad \text{mean}(y) \text{ returns } 0.5: \text{half of the values of } w_4 \text{ are } 4s.
\]

**Sampling from a finite population.** The sample function selects a sample of a specified size from a given population. For example, \(\text{sample}(1:365, 1)\) 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. When we used this function three times, we got the 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 \(\text{sample}(1:365, 25, \text{repl=}\text{T})\), 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.)

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 $P(\text{No Matches})$.

6. 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.

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 more difficult to find without simulation methods. Later in this article, we 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.

   \[ b \leftarrow \text{sample(1:365, 25, repl=T).} \]

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):

   \[ x \leftarrow 25 - \text{length(unique(b))}. \]

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 $X$s 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.)

The simulated value 0.4338 of the probability of seeing no matches is very near the known exact value 0.4313. Additional runs of the program consistently gave values of $E(X)$ in the range $0.80 \pm 0.02$.

7. Checking Assumptions. In modeling any real life situation, we must make assumptions. Here we hope 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 there are no leap years and 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.

Display 3

```r
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)
```

\[ > \text{mean(x)} \]

\[ [1] \ 0.8081 \]

\[ > \text{mean(x==0)} \]

\[ [1] \ 0.4338 \]

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. For example, we might 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).
```
Display 4

```r
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
```

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

```r
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.

Display 4 shows the minor change in the script of Display 3 that takes weights into account. (In R, equal weights are used if the `prob` parameter is omitted.) Based on this pattern of nonuniform birthdays, we obtain the values \( P(X = 0) \approx 0.42 \) and \( E(X) \approx 0.82 \), which are just barely distinguishable from the results we obtained under the assumption of equally likely birthdays.

Actual 1997–99 vital statistics for the US [4] show a little less variation in daily birth proportions than we assumed in Display 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.)

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

Empirical Daily Birth Proportions: By Month Jan '97 - Dec '99
(Percent of Uniform = 1/365 Per Day)

![Empirical Daily Birth Proportions](source: Nat’l Ctr. for Health Statistics)

The following vector of weights closely approximates the monthly birthrates averaged over these three years:

```r
w <- c(rep(96, 31), rep(98, 28), 25,
       rep(98, 61), rep(99, 31), rep(101, 30),
       rep(104, 31), rep(103, 31), rep(106, 30),
       rep(99, 31), rep(97, 30), rep(100, 31)).
```

Within the accuracy of our simulations (about two decimal places), the results using this vector are not easy to distinguish from results for uniformly distributed birthdays. From these and related simulations on birthday matching, we conclude that, although birthdays in the US are not actually uniformly distributed, it seems harmless in practice to assume they are.

Figure 4. Extreme Departures From Uniform. The horizontal axis shows numbers of weeks assumed to contain half of the annual birthdays. (See Section 8)

When one abandons the assumption that birthdays are equally likely, direct computation of \( P(\text{No Match}) \) is no longer elementary [5, 6]. But notice that 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.

8. Results for Extreme Departures From Uniform.

We have seen that neither the probability \( P(X = 0) \) of no matches nor the expected number \( E(X) \) of matches is much changed for mild departures from uniform. Now we explore briefly how these quantities change when the departure from uniform is great.

There are various ways to model departure from uniform. One that is easy to program is to assume that half of the annual birthdays are uniformly distributed over a period of \( k \) weeks, where \( k = 1, \ldots, 26 \), and that the other half are uniformly distributed over the rest of the year. Thus when \( k = 1 \), the departure from uniform is extreme, and when \( k = 26 \) the distribution is uniform.

The R code in Display 5 implements this idea for \( n = 25 \) people in the room, 365 days in a year, and \( m = 50,000 \) iterations for each value of \( k \). For example, if half of the population were born uniformly over
a period of $k = 4$ weeks and the rest had birthdays uniformly spread throughout the rest of the year, the weight vector $w$ would have 28 copies of the value 0.017830 and 337 copies of the value 0.001486 (upon scaling to total to unity). In this case, the results are $E(X) = 2.56$ and $P(X = 0) = 0.06$.

The curves in Figure 4 show the results from one run of the program. The horizontal reference lines in both parts of Figure 4 show values for the uniform case: specifically these are $E(X) = 0.80$, obtained from extensive simulations, and $P(X = 0) = 0.43$, obtained analytically in Section 2.

9. Student Projects. Here we suggest three student projects that provide practice in the use of R to simulate probability models. Additional instructional materials are available online [7].

Project 1: A poker hand consists of five cards drawn at random without replacement from a deck in which four cards are Aces. (a) What is the probability that no Aces will be drawn? (b) If $X$ is the number of Aces drawn, then what is $E(X)$? (c) What is the distribution of $X$? Draw its bar graph or histogram. Simulate all results.

All of these questions can be answered exactly using the hypergeometric distribution. The following code simulates approximate answers and gives exact ones:

```r
m <- 20000 # Hands simulated
n <- 5 # Cards per hand
x <- numeric(m) # Vector: Numb. Aces
for (i in 1:m) {
  h <- sample(1:52, n, repl=T) # ith Hand
  x[i] <- sum(h < 5)  # Aces in it
}
mean(x); var(x) # Exact: Both 1 for any n
```

Project 2: (a) A fair die is rolled $n = 6$ times. What is the probability all 6 faces are seen? [Answer: 6!/66.

(b) Repeat part (a) for $n = 10$. (c) Repeat parts (a) and (b) for a biased die with $P(1) = 1/12$, $P(2) = P(3) = 1/4$, $P(4) = P(5) = 1/6$ and $P(6) = 5/12$. Analytic solutions to most parts are difficult enough that simulation is the only way beginning students can get the answers. The R-code is very similar to that of the birthday problem except that the number of unique values is sought, not the number of matches. R-code for the biased version of part (b):

```r
m <- 30000; n <- 10; x <- numeric(m)
w <- c(1,2,2,2,3);
for (i in 1:m) {
  x[i] <- length(unique(sample(1:6, n, repl=T, prob=w)))
}
```

Project 3: A set of books with $n$ numbered volumes is arranged at random in a row. Find $E(X)$ and $V(X)$, where $X$ is the number of volumes in their correct numerical positions. [Answer: $E(X) = V(X) = 1$, for $n > 1$.]

For $n = 2$ or 3, argue from the sample space. For larger $n$, express $X$ as the sum of $n$ (dependent) indicator variables to get $E(X)$ easily; to get $V(X)$ is more advanced.

```
Display 5
```

```
par(mfrow=c(1,2))
plot(1:26, mn, xlab="Weeks", ylab="E(Number of Matches)", ylim=c(0,.5))
lines(c(1,26),c(.43,.43),col="red")
plot(1:26, pr, xlab="Weeks", ylab="P(No Matches)", ylim=c(0,.5))
lines(c(1,26),c(.43,.43),col="red")
```

References and Web Resources.

[1] R software and manuals are available from www.r-project.org
[6] Auxiliary instructional materials for this article are posted at www.science.ucdavis.edu/~btrumbo/bdmatch

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, esuess@csuhayward.edu.