# Chapter 10

# Simulation

Simulations provide a powerful way to answer questions and explore properties of statistical estimators and procedures. In this chapter, we will explore how to simulate data in a variety of common settings and apply some of the techniques introduced earlier.

## 10.1 Generating data

### 10.1.1 Generate categorical data

Simulation of data from continuous probability distributions is straightforward using the functions detailed in 3.1.1. Simulating from categorical distributions can be done manually or using some available functions.

```
data test;
p1 = .1; p2 = .2; p3 = .3;
do i = 1 to 10000;
   x = uniform(0);
   mycat1 = (x ge 0) + (x gt p1) + (x gt p1 + p2)
                     + (x gt p1 + p2 + p3);
   mycat2 = rantbl(0, .5, .4, .05);
   mycat3 = rand("TABLE", .3, .3, .4);
   output;
   end;
run;
```

```
proc freq data=test;
   tables mycat1 mycat2 mycat3;
run;

The FREQ Procedure
                                    Cumulative    Cumulative
mycat1    Frequency     Percent     Frequency      Percent
-----------------------------------------------------------
     1        1014       10.14          1014        10.14
     2        1966       19.66          2980        29.80
     3        2950       29.50          5930        59.30
     4        4070       40.70         10000       100.00

                                    Cumulative    Cumulative
mycat2    Frequency     Percent     Frequency      Percent
-----------------------------------------------------------
     1        5003       50.03          5003        50.03
     2        4064       40.64          9067        90.67
     3         481        4.81          9548        95.48
     4         452        4.52         10000       100.00

                                    Cumulative    Cumulative
mycat3    Frequency     Percent     Frequency      Percent
-----------------------------------------------------------
     1        2907       29.07          2907        29.07
     2        3017       30.17          5924        59.24
     3        4076       40.76         10000       100.00
```

The first argument to the `rantbl` function is the seed. The remaining arguments are the probabilities for the categories; if they sum to more than 1, the excess is ignored. If they sum to less than 1, the remainder is used for another category. The same is true for `rand("TABLE",...)`.

```
> options(digits=3)
> options(width=72) # narrow output
> p = c(.1,.2,.3)
> x = runif(10000)
> mycat1 = numeric(10000)
> for (i in 0:length(p)) {
      mycat1 = mycat1 + (x >= sum(p[0:i]))
      }
> table(mycat1)

mycat1
   1    2    3    4
 984 2035 3030 3951
```

```
> mycat2 = cut(runif(10000), c(0, 0.1, 0.3, 0.6, 1))
> summary(mycat2)

  (0,0.1] (0.1,0.3] (0.3,0.6]   (0.6,1]
      989      1999      3002      4010

> mycat3 = sample(1:4, 10000, rep=TRUE, prob=c(.1,.2,.3,.4))
> table(mycat3)

mycat3
   1    2    3    4
1013 2064 2970 3953
```

The `cut()` function (2.2.4) bins continuous data into categories with both endpoints defined by the arguments. Note that the `min()` and `max()` functions can be particularly useful here in the outer categories. The `sample()` function as shown treats the values $1, 2, 3, 4$ as a dataset and samples from the dataset 10,000 times with the probability of selection defined in the `prob` vector.

### 10.1.2 Generate data from a logistic regression

Here we show how to simulate data from a logistic regression (7.1.1). Our process is to generate the linear predictor, then apply the inverse link, and finally draw from a distribution with this parameter. This approach is useful in that it can easily be applied to other generalized linear models (7.1). Here we make the intercept $-1$, the slope 0.5, and generate $5,000$ observations.

```
data test;
intercept = -1;
beta = .5;
do i = 1 to 5000;
   xtest = normal(12345);
   linpred = intercept + (xtest * beta);
   prob = exp(linpred)/ (1 + exp(linpred));
   ytest = uniform(0) lt prob;
   output;
   end;
run;
```

Sometimes the voluminous SAS output can be useful, but here we just want to demonstrate that the parameter estimates are more or less accurate. The `ODS` system provides a way to choose only specific output elements.

```
ods select parameterestimates;
proc logistic data=test;
   model ytest(event='1') = xtest;
run;
ods select all;

The LOGISTIC Procedure
             Analysis of Maximum Likelihood Estimates

                                Standard          Wald
Parameter    DF    Estimate       Error    Chi-Square    Pr > ChiSq
Intercept     1     -1.0925      0.0338     1047.8259        <.0001
xtest         1      0.4978      0.0346      207.1199        <.0001
```

```
> intercept = -1
> beta = 0.5
> n = 5000
> xtest = rnorm(n, mean=1, sd=1)
> linpred = intercept + (xtest * beta)
> prob = exp(linpred)/(1 + exp(linpred))
> ytest = ifelse(runif(n) < prob, 1, 0)
```

While the `summary()` of a `glm` object is more concise than the default SAS output, we can display just the estimated values of the coefficients from the logistic regression model using the `coef()` function (see 6.4.1).

```
> coef(glm(ytest ~ xtest, family=binomial))

(Intercept)       xtest
     -1.018       0.479
```

### 10.1.3 Generate data from a generalized linear mixed model

In this example, we generate data from a generalized linear mixed model (7.4.7) with a dichotomous outcome. We generate 1500 clusters, denoted by `id`. There is one predictor with a common value for all observations in a cluster ($X_1$). Each observation within the cluster has an order indicator (denoted by $X_2$) which has a linear effect (`beta_2`), and there is an additional predictor which varies among observations ($X_3$). The dichotomous outcome $Y$ is generated from these predictors using a logistic link incorporating a normal distributed random intercept for each cluster.

```
data sim;
   sigbsq=4; beta0=-2; beta1=1.5; beta2=0.5; beta3=-1; n=1500;
   do i = 1 to n;
      x1 = (i lt (n+1)/2);
      randint = normal(0) * sqrt(sigbsq);
      do x2 = 1 to 3 by 1;
         x3 = uniform(0);
         linpred = beta0 + beta1*x1 + beta2*x2 + beta3*x3 + randint;
         expit = exp(linpred)/(1 + exp(linpred));
         y = (uniform(0) lt expit);
         output;
      end;
   end;
run;
```

This model can be fit using `proc nlmixed` (7.4.6) or `proc glimmix` (7.4.7). For large datasets, `proc nlmixed` (which uses numerical approximation to calculate the integral) can take a prohibitively long time to fit, and convergence can sometimes be problematic.

```
options ls=64;
ods select parameterestimates;

proc nlmixed data=sim qpoints=50;
   parms b0=1 b1=1 b2=1 b3=1;
   eta = b0 + b1*x1 + b2*x2 + b3*x3 + bi1;
   mu = exp(eta)/(1 + exp(eta));
   model y ~ binary(mu);
   random bi1 ~ normal(0, g11) subject=i;
   predict mu out=predmean;
run;
ods select all;

The NLMIXED Procedure
                    Parameter Estimates

                    Standard
Parameter  Estimate     Error    DF  t Value  Pr > |t|   Alpha
b0          -1.9946    0.1688  1499   -11.82    <.0001    0.05
b1           1.4866    0.1418  1499    10.49    <.0001    0.05
b2           0.5358   0.05153  1499    10.40    <.0001    0.05
b3          -0.9630    0.1596  1499    -6.04    <.0001    0.05
g11          4.1258    0.4064  1499    10.15    <.0001    0.05
            Parameter Estimates

Parameter     Lower       Upper    Gradient
b0          -2.3257     -1.6636    -0.00007
b1           1.2085      1.7648    0.000043
b2           0.4347      0.6369    -0.00006
b3          -1.2760     -0.6500    -4.63E-7
g11          3.3286      4.9231    -0.00001
```

On the other hand, `proc glimmix` frequently fails to reach convergence using the default maximization technique. We show below how to use a maximization technique that is often effective. We also show how to implement the Laplace approximation for the likelihood. This has better properties than the default pseudo-likelihood technique, but is not available for some more complex models.

```
ods select parameterestimates covparms;

proc glimmix data=sim order=data method=laplace;
   nloptions maxiter=100 technique=dbldog;
   model y = x1 x2 x3 / solution dist=bin;
   random int / subject=i;
run;

ods select all;

The GLIMMIX Procedure
       Covariance Parameter Estimates

                                    Standard
Cov Parm     Subject    Estimate       Error
Intercept    i            3.3195      0.3317

                  Solutions for Fixed Effects

                          Standard
Effect       Estimate       Error       DF    t Value    Pr > |t|
Intercept     -1.9500      0.1634     1498     -11.93      <.0001
x1             1.4567      0.1332     2998      10.93      <.0001
x2             0.5197     0.05060     2998      10.27      <.0001
x3            -0.9317      0.1553     2998      -6.00      <.0001
```

Discrepancies between the two sets of estimates arise mainly from the differences between the numeric integration in `proc nlmixed` and the use of the Laplace approximation in `proc glimmix`.

The R simulation uses the approach introduced in 4.1.3 applied to a more complex setting, with each of the components built up part by part.

```
> n = 1500; p = 3; sigbsq = 4
> beta = c(-2, 1.5, 0.5, -1)
> id = rep(1:n, each=p)         # 1 1 ... 1 2 2 ... 2 ... n
> x1 = as.numeric(id < (n+1)/2) # 1 1 ... 1 0 0 ... 0
> randint = rep(rnorm(n, 0, sqrt(sigbsq)), each=p)
> x2 = rep(1:p, n)              # 1 2 ... p 1 2 ... p ...
> x3 = runif(p*n)
> linpred = beta[1] + beta[2]*x1 + beta[3]*x2 + beta[4]*x3 + randint
> expit = exp(linpred)/(1 + exp(linpred))
> y = runif(p*n) < expit        # generate a logical as our outcome
```

We fit the model using the `glmer()` function from the `lme4` package.

```
> library(lme4)
> glmmres = glmer(y ~ x1 + x2 + x3 + (1|id), family=binomial(link="logit"))
> summary(glmmres)

Generalized linear mixed model fit by maximum likelihood ['glmerMod']
 Family: binomial ( logit )
Formula: y ~ x1 + x2 + x3 + (1 | id)

     AIC      BIC   logLik deviance
    5323     5355    -2657     5313

Random effects:
 Groups Name        Variance Std.Dev.
 id     (Intercept) 2.6      1.61
Number of obs: 4500, groups: id, 1500

Fixed effects:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -2.0118     0.1406   -14.3  < 2e-16 ***
x1            1.4084     0.1127    12.5  < 2e-16 ***
x2            0.4640     0.0451    10.3  < 2e-16 ***
x3           -0.8168     0.1409    -5.8  6.8e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
   (Intr) x1     x2
x1 -0.436
x2 -0.665  0.050
x3 -0.440 -0.052 -0.035
```

### 10.1.4 Generate correlated binary data

Another way to generate correlated dichotomous outcomes $Y_1$ and $Y_2$ is based on the probabilities corresponding to the $2 \times 2$ table. Given these cell probabilities, the variable probabilities can be expressed as a function of the marginal probabilities and the desired correlation, using the methods of Lipsitz and colleagues [107]. Here we generate a sample of 1000 values where: $P(Y_1 = 1) = .15, P(Y_2 = 1) = .25$, and $\text{Corr}(Y_1, Y_2) = 0.40$.

```
data test;
   p1=.15; p2=.25; corr=0.4;
   p1p2=corr*sqrt(p1*(1-p1)*p2*(1-p2)) + p1*p2;
   do i = 1 to 10000;
      cat=rand('TABLE', 1-p1-p2+p1p2, p1-p1p2, p2-p1p2);
      y1=0;
      y2=0;
      if cat=2 then y1=1;
      else if cat=3 then y2=1;
      else if cat=4 then do;
         y1=1;
         y2=1;
      end;
      output;
   end;
run;
```

```
> p1 = .15; p2 = .25; corr = 0.4; n = 10000
> p1p2 = corr*sqrt(p1*(1-p1)*p2*(1-p2)) + p1*p2
> library(Hmisc)
> vals = rMultinom(matrix(c(1-p1-p2+p1p2, p1-p1p2, p2-p1p2, p1p2),
     nrow=1, ncol=4), n)
> y1 = rep(0, n); y2 = rep(0, n)   # put zeroes everywhere
> y1[vals==2 | vals==4] = 1         # and replace them with ones
> y2[vals==3 | vals==4] = 1         # where needed
> rm(vals, p1, p2, p1p2, corr, n)    # cleanup
```

The generated data is close to the desired values.

```
options ls = 68;
proc corr data=test;
   var y1 y2;
run;

The CORR Procedure
   2  Variables:    y1       y2
                      Simple Statistics
Variable           N          Mean       Std Dev           Sum
y1             10000       0.14630       0.35342          1463
y2             10000       0.24550       0.43040          2455
         Simple Statistics

Variable       Minimum       Maximum
y1                   0       1.00000
y2                   0       1.00000

Pearson Correlation Coefficients, N = 10000
         Prob > |r| under H0: Rho=0

              y1            y2
y1       1.00000       0.38451
                        <.0001
y2       0.38451       1.00000
          <.0001
```

```
> cor(y1, y2)

[1] 0.412

> table(y1)

y1
   0    1
8476 1524

> table(y2)

y2
   0    1
7398 2602
```

### 10.1.5 Generate data from a Cox model

To simulate data from a Cox proportional hazards model (7.5.1), we need to model the hazard functions for both time to event and time to censoring. In this example, we use a constant baseline hazard, but this can be modified by specifying other `scale` parameters for the Weibull random variables.

```
data simcox;
   beta1 = 2;
   beta2 = -1;
   lambdat = 0.002; *baseline hazard;
   lambdac = 0.004; *censoring hazard;
   do i = 1 to 10000;
      x1 = normal(0);
      x2 = normal(0);
      linpred = exp(-beta1*x1 - beta2*x2);
      t = rand("WEIBULL", 1, lambdaT * linpred);
      * time of event;
      c = rand("WEIBULL", 1, lambdaC);
      * time of censoring;
      time = min(t, c);    * time of first?;
      censored = (c lt t); * 1 if censored;
      output;
   end;
run;
```

```
> # generate data from Cox model
> n = 10000
> beta1 = 2; beta2 = -1
> lambdaT = .002        # baseline hazard
> lambdaC = .004  # hazard of censoring
> x1 = rnorm(n)   # standard normal
> x2 = rnorm(n)
> # true event time
> T = rweibull(n, shape=1, scale=lambdaT*exp(-beta1*x1-beta2*x2))
> C = rweibull(n, shape=1, scale=lambdaC)   #censoring time
> time = pmin(T,C)  #observed time is min of censored and true
> censored = (time==C)   # set to 1 if event is censored
> # fit Cox model
> library(survival)
> survobj = coxph(Surv(time, (1-censored))~ x1 + x2, method="breslow")
```

These parameters generate data where approximately 40% of the observations are censored.

Note that `proc phreg` and `coxph()` expect different things: a censoring indicator and an observed event indicator, respectively. Here we made a censoring indicator in both simulations, though this leads to the somewhat awkward syntax shown in the `coxph()` function.

The `phreg` procedure (7.5.1) will describe the censoring patterns as well as the results of fitting the regression model.

```
options ls = 68;
ods select censoredsummary parameterestimates;
proc phreg data=simcox;
   model time*censored(1) = x1 x2;
run;

The PHREG Procedure
Summary of the Number of Event and Censored Values

                                    Percent
   Total       Event    Censored    Censored
   10000        5999        4001       40.01

            Analysis of Maximum Likelihood Estimates

              Parameter   Standard                          Hazard
Parameter DF   Estimate      Error Chi-Square Pr > ChiSq     Ratio
x1         1    1.99434    0.02230  7997.1857     <.0001     7.347
x2         1   -0.98394    0.01563  3962.9682     <.0001     0.374
```

In R we tabulate the censoring indicator, then display the results as well as the associated 95% confidence intervals.

```
> table(censored)

censored
FALSE  TRUE
 5968  4032

> print(survobj)

Call:
coxph(formula = Surv(time, (1 - censored)) ~ x1 + x2, method = "breslow")


     coef exp(coef) se(coef)     z p
x1  2.006     7.433   0.0222  90.4 0
x2 -0.988     0.373   0.0157 -62.7 0

Likelihood ratio test=11490  on 2 df, p=0  n= 10000, number of events= 5968

> confint(survobj)

   2.5 % 97.5 %
x1  1.96  2.049
x2 -1.02 -0.957
```

The results are similar to the true parameter values.