Simulating datasets

Ken Kellner

September 10, 2021

1 Outline

2 Introduction

Simulating datasets is a powerful and varied tool when conducting `unmarked` analyses. Writing our own code to simulate a dataset based on a given model is an excellent learning tool, and can help us test if a given model is generating the expected results. If we simulate a series of datasets based on a fitted model, and calculate a statistic from each of those fits, we can generate a distribution of the statistic - this is what the `parboot` function does. This can be helpful, for example, when testing goodness of fit. Finally, simulation can be a useful component of power analysis when a closed-form equation for power is not available.

`unmarked` provides two different ways of generating simulated datasets, depending on the stage we are at in the modeling process.

1. Generating simulated datasets from a fitted model we already have
2. Generating simulated datasets from scratch

For (1), we simply call the `simulate` method on our fitted model object and new dataset(s) are generated. This is the approach used by `parboot`. In this vignette we will focus on (2), a more flexible approach to simulation, also using the `simulate` method, that allows us to generate a dataset corresponding to any `unmarked` model from scratch.

3 Components of a call to simulate

We will need to provide, at a minimum, four pieces of information to `simulate` in order to simulate a dataset from scratch in `unmarked`.

1. The name of the fitting function for the model we want to simulate from, as a character string
2. A list of formulas, one per submodel, containing the names of the covariates we want to include in each
3. A list of vectors of regression coefficients (intercepts and slopes), one per submodel, matching the formulas
4. A list of design components; for example, the number of sites and number of observations per site

A number of other arguments are available, e.g. for how to customize how the covariates are randomly generated or for distributions to use when simulating abundances. We'll show those later. The easiest way to demonstrate how to use `simulate` is to look at an example: we'll start with a simple one for occupancy.

4 Simulating an occupancy dataset

Suppose we want to simulate an occupancy dataset in which site occupancy is affected by elevation. The first piece of information needed is the name of model to use: the fitting function for occupancy is `occu`, so the first argument to `simulate` and the name of the model will be `"occu"`.

4.1 Formulas

Second we must define the desired model structure as a list of formulas, one per submodel. ”Submodels” here are the hierarchical components of the model; for example, an occupancy model has a state (occupancy) submodel and an observation (detection) submodel. These submodels are identified by short names: `state` and `det`. We will use these short names repeatedly. In order to identify which submodels are needed and what their short names are, we can simply fit any model of that type (e.g. from the example) and call `names(model)`.

```
> library(unmarked)
> umf <- unmarkedFrameOccu(y=matrix(c(0,1,0,1,1,0,0,0,1), nrow=3))
> mod <- occu(~1~1, umf)
> names(mod)
[1] "state" "det"
```

Formulas are supplied as a named list. The list has one element per submodel, and the names of the elements are the short names defined above. Each list element is a formula, containing the desired number of covariates to use, and the names of these covariates. Below we define our list of formulas, including an effect of elevation on occupancy (note we could name this whatever we want, here we call it `elev`). We don’t want any covariates on detection probability, so the formula defines the model as intercept only: `1`.

```
> forms <- list(state=~elev, det=~1)
```

4.2 Regression coefficients

Next we must tell `unmarked` what the values for the intercept and regression coefficients in each submodel should be. Once again, this is a named list, one element for each submodel. Each list element is a numeric vector. The components of each numeric vector must also be named, matching the covariate names in our list of formulas. Don’t forget we also must specify a value for the intercept in each submodel (can be named `Intercept` or `intercept`). If we are not sure exactly how to structure this list, just skip it for now: `unmarked` can generate a template for us to fill in later.

```
> coefs <- list(state=c(intercept=0, elev=-0.4), det=c(intercept=0))
```

We have a list with two elements, each a numeric vector. Both contain intercept values, and the `state` vector also contains a value corresponding to the desired effect of our covariate `elev`.

4.3 Study design information

Finally, we need to give `unmarked` information about the study design. This is pretty simple: we just need a list containing values for `M`, the number of sites, and `J` the number of surveys per site. For models with multiple primary periods, we’d also need a value of `T`, the number of primary periods.

```
> design <- list(M=300, J=8) # 300 sites, 8 occasions per site
```

4.4 Put it all together

We’re now ready to simulate a dataset. To do this we use the `simulate` function, providing as arguments the name of the model `"occu"` and the three lists we constructed above. Actually, first, let’s not supply the `coefs` list, to show how `unmarked` will generate a template for us to use:

```
> simulate("occu", formulas=forms, design=design)
```

```
coefs argument should be a named list of named vectors, with the following structure
        (replacing 0s with your desired coefficient values):

$state
intercept      elev
        0         0

$det
intercept
        0
```

We can replicate this provided list structure and fill in our own numeric values. Once we have our coefficients set up properly, add them to the function call:

```
> occu_umf <- simulate("occu", formulas=forms, coefs=coefs, design=design)
> head(occu_umf)
Data frame representation of unmarkedFrame object.
   y.1 y.2 y.3 y.4 y.5 y.6 y.7 y.8       elev
1    0   0   0   0   0   0   0   0 -0.7152422
2    0   0   0   0   0   0   0   0 -0.7526890
3    0   0   0   0   1   0   1   0 -0.9385387
4    0   0   0   0   0   0   0   0 -1.0525133
5    1   0   0   0   0   0   1   0 -0.4371595
6    0   1   0   1   1   0   0   0  0.3311792
7    1   1   1   0   0   0   0   0 -2.0142105
8    0   0   0   0   0   0   0   0  0.2119804
9    1   0   0   1   0   1   0   0  1.2366750
10   0   0   0   0   0   0   0   0  2.0375740
```

`unmarked` has generated a presence-absence dataset as well as values for covariate `elev`. We can check that it worked as expected by fitting the corresponding model to the dataset, and making sure the estimated values are similar:

```
> (occu(~1 ~elev, occu_umf))
Call:
occu(formula = ~1 ~ elev, data = occu_umf)

Occupancy:
            Estimate    SE      z  P(>|z|)
(Intercept)  -0.0845 0.119 -0.712 0.476492
elev         -0.4407 0.125 -3.514 0.000442

Detection:
 Estimate   SE       z P(>|z|)
 -0.00451 0.06 -0.0751    0.94

AIC: 1992.853
```

4.5 Customizing the covariates

By default, a covariate will be continuous and come from a standard normal distribution (mean 0, SD 1). However, we can control this using the `guide` argument. For example, suppose we want elevation to come from a random normal, but with a mean of 2 and a standard deviation of 0.5. We can provide a named list to the `guide` argument as follows:

```
> guide <- list(elev=list(dist=rnorm, mean=2, sd=0.5))
```

`guide` contains one element, called `elev`, which is also a list and contains three components:

1. The random distribution function to use, `rnorm`
2. The mean of the distribution
3. The SD of the distribution

```
> occu_umf <- simulate("occu", formulas=forms, coefs=coefs, design=design, guide=guide)
> head(occu_umf)
Data frame representation of unmarkedFrame object.
   y.1 y.2 y.3 y.4 y.5 y.6 y.7 y.8     elev
1    0   0   0   0   0   0   0   0 2.063074
2    0   0   0   0   0   0   0   0 2.236400
3    0   0   0   0   0   0   0   0 1.829623
4    0   0   0   0   0   0   0   0 1.879105
5    0   0   0   0   0   0   0   0 2.689377
6    0   0   0   0   0   0   0   0 1.830558
7    0   0   0   0   0   0   0   0 2.010068
8    0   0   0   0   0   0   0   0 2.188481
9    1   1   1   0   0   0   1   1 1.784138
10   0   0   0   0   0   0   0   0 2.979532
```

You can see the `elev` covariate now has values corresponding to the desired distribution. Note that the elements of the list will depend on the arguments required by the random distribution function. For example, to use a uniform distribution instead:

```
> guide <- list(elev=list(dist=runif, min=0, max=1))
> occu_umf <- simulate("occu", formulas=forms, coefs=coefs, design=design, guide=guide)
> head(occu_umf)
Data frame representation of unmarkedFrame object.
   y.1 y.2 y.3 y.4 y.5 y.6 y.7 y.8      elev
1    0   0   0   0   0   0   0   0 0.4154205
2    0   0   1   1   0   1   1   0 0.1033568
3    1   1   1   0   0   0   1   1 0.8845174
4    1   0   0   1   0   0   1   0 0.9871606
5    0   0   0   1   0   0   1   1 0.5916545
6    0   1   0   0   1   1   1   1 0.9046272
7    0   0   0   0   0   0   0   0 0.3075379
8    0   0   0   0   0   0   0   0 0.6759720
9    0   0   0   0   0   0   0   0 0.7358022
10   0   0   0   0   0   0   0   0 0.4075122
```

It is also possible to define a categorical (factor) covariate. We specify an entry in the `guide` list, but instead of a list, we supply a call to `factor` which defines the desired factor levels. For example, suppose we want to add a new `landcover` covariate to our simulated model. First, define the new formulas:

```
> forms2 <- list(state=~elev+landcover, det=~1)
```

And then the new guide, including the information about factor levels:

```
> guide <- list(landcover=factor(levels=c("forest","grass","urban")))
```

We'd also need an updated `coefs` since we have a new covariate. Defining the `coefs` when you have factors in your model is a little trickier, since R names the effects as a combination of the factor name and the level name. There is no coefficient for the reference level (`"forest"` in our example), but we need to provide coefficients for both `"grass"` and `"urban"`. When combined with the factor name the complete coefficient names for these two will be `landcovergrass` and `landcoverurban`. The easiest way to make sure we get these names right is to let `unmarked` generate a template `coefs` for you as shown above, and then fill it in.

```
> # forest is the reference level for landcover since it was listed first
> coefs2 <- list(state=c(intercept=0, elev=-0.4, landcovergrass=0.2,
                       landcoverurban=-0.7), det=c(intercept=0))
```

```
> head(simulate("occu", formulas=forms2, coefs=coefs2, design=design, guide=guide))
Data frame representation of unmarkedFrame object.
   y.1 y.2 y.3 y.4 y.5 y.6 y.7 y.8         elev landcover
1    0   1   1   1   0   1   0   1  0.253522341     grass
2    1   1   0   0   0   0   1   0  0.243522140    forest
3    0   1   0   0   1   0   1   0  0.719590250     grass
4    0   1   1   1   1   0   0   1  0.772732980     grass
5    0   0   0   0   0   0   0   0 -0.008522864    forest
6    1   1   0   1   1   1   1   0 -0.389069155     urban
7    0   0   0   0   0   0   0   0  1.328233354     urban
8    0   0   1   1   1   1   0   1  0.053012552    forest
9    1   0   0   1   0   1   1   1 -0.376265158     grass
10   1   1   1   0   1   0   0   1 -0.946238347     grass
```

Our output dataset now includes a new categorical covariate.

4.6 Models that require more information

More complex models might require more information for simulation. Nearly any argument provided to either the fitting function for the model, or the corresponding `unmarkedFrame` constructor, can be provided as an optional argument to `simulate` to customize the simulation. For example, we may want to specify that abundance should be simulated as a negative binomial, instead of a Poisson, for `pcount`. This information is simply added as additional arguments to `simulate`. For example, we can simulate a `pcount` dataset using the negative binomial (`"NB"`) distribution. The negative binomial has an additional parameter to estimate (`alpha`) so we must also add an element to `coefs`.

```
> coefs$alpha <- c(alpha=0.5)
> head(simulate("pcount", formulas=forms, coefs=coefs, design=design, mixture="NB"))
Data frame representation of unmarkedFrame object.
   y.1 y.2 y.3 y.4 y.5 y.6 y.7 y.8        elev
1    0   0   0   0   0   0   0   0 -0.37071853
2    1   2   2   2   0   0   2   1  0.15751357
3    0   0   0   0   0   0   0   0 -0.98832179
4    1   0   1   0   0   1   0   0  0.30113025
5    0   0   0   0   0   0   0   0 -1.47189644
6    0   0   0   0   0   0   0   0  0.02303002
7    2   3   1   2   3   3   3   2  1.30191316
8    0   0   0   0   0   0   0   0 -0.91494417
9    0   0   0   0   0   0   0   0  0.21043315
10   1   1   1   1   2   1   0   1 -2.00025076
```

In the next section we will show a more detailed example involving these additional arguments.

5 Simulating a more complex dataset: gdistremoval

The `gdistremoval` function fits the model of Amundson et al. (2014), which estimates abundance using a combination of distance sampling and removal sampling data. When simulating a dataset based on this model, we have to provide several additional pieces of information related to the structure of the distance and removal sampling analyses.

To begin, we will define the list of formulas. A `gdistremoval` model, when there is only one primary period, has three submodels: abundance (`"lambda"`), distance sampling (`"dist"`), and removal sampling (`"rem"`). We will fit a model with an effect of elevation `elev` on abundance and an effect of wind `wind` on removal probability.

```
> forms <- list(lambda=~elev, dist=~1, rem=~wind)
```

Next we will define the corresponding coefficients. We will set mean abundance at 5. The intercept is on the log scale, thus the intercept for `lambda` will be `log(5)`. The scale parameter for the detection function will be 50, and again it is on the log scale. The intercept for the removal probability is on the logit scale, so we will set the intercept at -1 (equivalent to a mean removal probability of about 0.27). Don't forget the covariate effects on `lambda` and removal.

```
> coefs <- list(lambda=c(intercept=log(5), elev=0.7),
                dist=c(intercept=log(50)), rem=c(intercept=-1, wind=-0.3))
```

Our study will have 300 sites. This model is unique in that we have to specify the number of two different types of observations: (1) the number of distance sampling bins (`Jdist`), and the number of removal intervals (`Jrem`).

```
> design <- list(M = 300, Jdist=4, Jrem=5)
```

Finally we are ready to simulate the dataset. In addition to the name of the model, `forms`, `coefs` and `design`, we also need to provide some additional information. We need to define the distance breaks for the distance sampling part of the model (there should be `Jdist+1` of these), and also the key function to use when simulating the detection process.

```
> umf <- simulate("gdistremoval", formulas=forms, coefs=coefs, design=design,
                  dist.breaks=c(0,25,50,75,100), keyfun="halfnorm", unitsIn="m")
> head(umf)
Data frame representation of unmarkedFrame object.
   yDist.1 yDist.2 yDist.3 yDist.4 yRem.1 yRem.2 yRem.3 yRem.4 yRem.5
1        4       2      10       2      4      7      6      1      0
2        0       0       0       0      0      0      0      0      0
3        0       1       2       0      0      1      1      0      1
4        4       1       8       1      4      1      4      3      2
5        0       1       5       2      3      3      0      2      0
6        0       1       2       4      3      1      1      2      0
7        2       3       3       2      2      3      2      3      0
8        1       1       0       0      1      0      1      0      0
9        0       0       1       2      2      1      0      0      0
10       0       2       2       1      2      0      2      1      0
          elev     wind.1       wind.2     wind.3     wind.4
1   1.61334622  1.3649476 -1.127500269 -0.1558707 -2.1302728
2  -0.80567600  0.3281250  0.598608134  0.7481760 -0.4382321
3  -1.14049439  1.5145357  0.310432089 -0.3901116 -0.7034511
4   1.46155447 -0.1052677  0.855721989 -0.8086349 -0.8053908
5   0.72496346  0.8454980 -1.340275756 -0.3633820  0.2957100
6   0.78390528 -0.1319023 -0.007654917 -0.7639284  0.2234584
7   1.08265721  0.9492643  0.980460084 -0.9523599 -1.2450675
8  -0.41839316  1.1423154  0.961612070  0.4852678  1.2666489
9  -1.02443597 -1.7832070 -1.339807559  0.6682450 -0.3279624
10  0.01774842  0.8019722  1.877960654  0.5301769 -1.7808959
        wind.5
1   0.04600723
2  -1.43481532
3   0.68665606
4   0.02402033
5  -0.55941853
6   1.29132573
7   0.41913508
8   0.57672752
9  -1.31624876
10  0.36150801
```

The result is a dataset containing a combination of distance, removal, and covariate data. We can check to see if fitting a model to this dataset recovers our specified coefficient values:

```
> (fit <- gdistremoval(lambdaformula=~elev, removalformula=~wind,
                     distanceformula=~1, data=umf))
Call:
gdistremoval(lambdaformula = ~elev, removalformula = ~wind, distanceformula = ~1,
    data = umf)

Abundance:
            Estimate     SE    z   P(>|z|)
(Intercept)     2.67 0.0410 65.2  0.00e+00
elev            0.67 0.0203 33.0 1.21e-238

Distance:
 Estimate     SE   z P(>|z|)
        4 0.0223 179       0

Removal:
            Estimate     SE      z  P(>|z|)
(Intercept)   -0.962 0.0520 -18.50 1.90e-76
wind          -0.262 0.0272  -9.63 5.86e-22

AIC: 6209.99
```

Looks good.

6 Conclusion

The `simulate` function provides a flexible tool for simulating data from any model in `unmarked`. These datasets can be used for a variety of purposes, such as for teaching examples, testing models, or developing new tools that work with `unmarked`. Additionally, simulating datasets is a key component of the power analysis workflow in `unmarked` - see the power analysis vignette for more examples.

References

Amundson, C. L., J. A. Royle, and C. M. Handel, 2014. A hierarchical model combining distance sampling and time removal to estimate detection probability during avian point counts. *The Auk* **131**:476–494.