Discrete Response Simulations

Vijay Ivaturi, Chris Rackauckas

July 19th, 2019

1 Introduction

In this tutorial we will go over the simulation of discrete responses. Many pharmacometrics scenarios have observables, such as pain scores or counts, which necessarily have to be discrete. Handling this discreteness can be paramount to getting an appropriate data fit and to properly understand the variation.

Luckily, in Pumas, discrete outputs are handled no differently from the rest of the Pumas toolchain. In Pumas, to have a discrete distribution as output, simply have that your `derived` or `observed` variables come from a discrete distribution like a `Poisson` process.

1.1 Binary Response Example

First, let's take a look at a binary response. A binary response is a model which gives an output of 0 or 1 with a probability `p`. In Pumas, this is represented by a `Bernoulli(p)` distribution.

To get started, first let's load `Pumas` and read in some example data:

```
using Pumas, StatsFuns
data = read_pumas(example_nmtran_data("pain_remed"),
    cvs = [:arm, :dose, :conc, :painord,:remed],event_data=false)

Population
  Subjects: 160
  Covariates: arm, dose, conc, painord, remed
  Observables: dv
```

Next let's implement a model with a binary response. Here, we do not have an `@dynamics` porition. Pumas will automatically handle this. In our `derived`, we define a `logistic` from StatsFuns

```
import StatsFuns.logistic
binary_model = @model begin
    @param begin
        intercept ∈ RealDomain(init=0.001)
        tvslope ∈ RealDomain(init=0.0001)
        Ω ∈ VectorDomain(1)
    end

    @random begin
```

```
        η ~ MvNormal(Ω)
    end

    @covariates arm dose

    @pre begin
        rx = dose > 0 ? 1 : 0
        slope = tvslope*rx
        logit = intercept + slope + η[1]
    end

    @derived begin
        pain = logistic(logit)
        dv ~ Bernoulli(logistic(logit))
    end
end

PumasModel
  Parameters: intercept, tvslope, Ω
  Random effects: η
  Covariates: arm, dose
  Dynamical variables:
  Derived: pain, dv
  Observed: pain, dv
```

Note that we could have alternatively defined our `@pre` like:

```
@pre begin
    logit = intercept + tvslope*(dose > 0 ? 1 : 0) + η[1]
    logit = intercept + tvslope*Int(dose > 0) + η[1]
end
```

more directly instead of using `logistic`.

Now let's fit our model to the data:

```
param = (
    intercept = 0.001,
    tvslope = 0.0001,
    Ω = [1.0]
    )
res = fit(binary_model,data,param,Pumas.LaplaceI())

FittedPumasModel

Successful minimization:                true

Likelihood approximation:     Pumas.LaplaceI
Objective function value:            1083.01
Total number of observation records:    1920
Number of active observation records:   1920
Number of subjects:                      160

-------------------
          Estimate
-------------------
intercept -1.3085
tvslope    1.7386
Ω_1         1.5374
-------------------
```

and simulate some outputs:

```
sim = simobs(binary_model,data,res.param)
simdf = DataFrame(sim, include_events=false)
```

	id	time	pain	dv	arm	dose	conc	painord	remed
	String	Float64	Float64	Bool	Int64	Int64	Float64	Int64	Int64
1	1	0.0	0.362518	false	2	20	0.0	3	0
2	1	0.5	0.362518	false	2	20	1.15578	1	0
3	1	1.0	0.362518	false	2	20	1.37211	0	0
4	1	1.5	0.362518	false	2	20	1.30058	0	0
5	1	2.0	0.362518	false	2	20	1.19195	1	0
6	1	2.5	0.362518	false	2	20	1.13602	1	0
7	1	3.0	0.362518	false	2	20	0.873224	0	0
8	1	4.0	0.362518	false	2	20	0.739963	1	0
9	1	5.0	0.362518	false	2	20	0.600143	2	0
10	1	6.0	0.362518	false	2	20	0.425624	1	0
11	1	7.0	0.362518	false	2	20	0.363418	1	0
12	1	8.0	0.362518	false	2	20	0.304177	1	0
13	2	0.0	0.394536	false	3	80	0.0	3	0
14	2	0.5	0.394536	false	3	80	4.93492	1	0
15	2	1.0	0.394536	false	3	80	4.56849	0	0
16	2	1.5	0.394536	false	3	80	4.23436	0	0
17	2	2.0	0.394536	false	3	80	3.35444	0	0
18	2	2.5	0.394536	false	3	80	2.70046	1	0
19	2	3.0	0.394536	false	3	80	2.38586	1	0
20	2	4.0	0.394536	false	3	80	1.84095	1	0
21	2	5.0	0.394536	false	3	80	1.64535	1	0
22	2	6.0	0.394536	false	3	80	1.36486	0	0
23	2	7.0	0.394536	false	3	80	1.20802	1	0
24	2	8.0	0.394536	false	3	80	0.983679	0	0
25	3	0.0	0.185952	false	0	0	0.0	3	0
26	3	0.5	0.185952	false	0	0	0.0	1	0
27	3	1.0	0.185952	false	0	0	0.0	1	0
28	3	1.5	0.185952	false	0	0	0.0	2	0
29	3	2.0	0.185952	false	0	0	0.0	1	0
30	3	2.5	0.185952	false	0	0	0.0	1	0
31	3	3.0	0.185952	false	0	0	0.0	2	0
32	3	4.0	0.185952	false	0	0	0.0	2	0
33	3	5.0	0.185952	false	0	0	0.0	2	0
34	3	6.0	0.185952	false	0	0	0.0	2	0
35	3	7.0	0.185952	false	0	0	0.0	2	0
36	3	8.0	0.185952	false	0	0	0.0	2	0
37	4	0.0	0.435142	true	0	0	0.0	3	0
38	4	0.5	0.435142	true	0	0	0.0	2	0
39	4	1.0	0.435142	true	0	0	0.0	2	0
40	4	1.5	0.435142	true	0	0	0.0	1	0
41	4	2.0	0.435142	true	0	0	0.0	1	0
42	4	2.5	0.435142	true	0	0	0.0	2	0
43	4	3.0	0.435142	true	0	0	0.0	2	0
44	4	4.0	0.435142	true	0	0	0.0	1	0
45	4	5.0	0.435142	true	0	0	0.0	1	0
46	4	6.0	0.435142	true	0	0	0.0	2	0
47	4	7.0	0.435142	true	0	0	0.0	2	0
48	4	8.0	0.435142	true	0	0	0.0	1	0
49	5	0.0	0.510977	true	3	80	0.0	3	0
50	5	0.5	0.510977	true	3	80	4.98932	2	0
51	5	1.0	0.510977	true	3	80	4.41742	1	0
52	5	1.5	0.510977	true	3	80	3.82287	1	0

Note that now our simulation output for `dv` is true/false values pulled with probability given by `logit` dependent on the individual's random effects.

1.2 Poisson Response Example

Next let's use a `Poisson` counting process in our model. Here we generate a population where everyone is receiving the same doses as a covariate.

```
pop = Population(map(i -> Subject(id=i,cvs=(dose = i.*10,),time=[0.0]),1:10))
```

```
Population
  Subjects: 10
  Covariates: dose
```

Now we define our model without dynamics, and directly use the dose information to predict the count for some observable `dv`:

```
poisson_model = @model begin
  @param begin
    tvbase ∈ RealDomain(init=3.0, lower=0.1)
    d50 ∈ RealDomain(init=50, lower=0.1)
    Ω  ∈ PSDDomain(fill(0.1, 1, 1))
  end

  @random begin
    η ~ MvNormal(Ω)
  end

  @pre begin
    baseline = tvbase*exp(η[1])
  end

  @covariates dose

  @derived begin
    dv ~ @. Poisson(baseline*(1-dose/(dose + d50)))
  end
end
```

```
PumasModel
  Parameters: tvbase, d50, Ω
  Random effects: η
  Covariates: dose
  Dynamical variables:
  Derived: dv
  Observed: dv
```

and simulate runs from the model:

```
sim = simobs(poisson_model,pop)
simdf = DataFrame(sim, include_events=false)
```

	id	time	dv	dose
	String	Float64	Int64	Int64
1	1	0.0	1	10
2	2	0.0	2	20
3	3	0.0	1	30
4	4	0.0	2	40
5	5	0.0	0	50
6	6	0.0	3	60
7	7	0.0	0	70
8	8	0.0	1	80
9	9	0.0	2	90
10	10	0.0	0	100

Here `dv` is an integer output probabilistically dependent on the dose.