A Multiplicative Poisson Model

A simple way to write the multiplicative model is to re-express the data in the form of a regression of the outcome $y$ on a dummy variable $x$ which is 1 if the person was in the experimental group and 0 if he/she was in the control group:

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>$\cdots$</th>
<th>287</th>
<th>288</th>
<th>289</th>
<th>$\cdots$</th>
<th>572</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i$</td>
<td>0</td>
<td>0</td>
<td>$\cdots$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$\cdots$</td>
<td>1</td>
</tr>
<tr>
<td>$y_i$</td>
<td>1</td>
<td>0</td>
<td>$\cdots$</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>$\cdots$</td>
<td>1</td>
</tr>
</tbody>
</table>

Then for $i = 1, \ldots, n = 572$ the multiplicative model can be written

$$ (y_i | \lambda_i) \overset{\text{indep}}{\sim} \text{Poisson}(\lambda_i) $$

$$ \log(\lambda_i) = \gamma_0 + \gamma_1 x_i \quad (4) $$

$$ (\gamma_0, \gamma_1) \sim \text{diffuse} $$

In this model the control people have

$$ \log(\lambda_i) = \gamma_0 + \gamma_1(0) = \gamma_0, \quad \text{i.e.,} \quad \lambda_C = e^{\gamma_0}, \quad (5) $$

and the experimental people have

$$ \log(\lambda_i) = \gamma_0 + \gamma_1(1) = \gamma_0 + \gamma_1, \quad \text{i.e.,} \quad \lambda_E = e^{\gamma_0 + \gamma_1} = e^{\gamma_0} e^{\gamma_1} = \lambda_C e^{\gamma_1}. \quad (6) $$

Now you may remember from basic Taylor series that for $\gamma_1$ not too far from 0

$$ e^{\gamma_1} \approx 1 + \gamma_1, \quad (7) $$
A Multiplicative Poisson Model

so that finally (for $\gamma_1$ fairly near 0)

$$\lambda_E \doteq (1 + \gamma_1) \lambda_C,$$

which is a way of expressing equation (3) in Poisson language.

Fitting this model in classicBUGS is easy:

```R
model poisson2;

cost

n = 572;

var

gamma.0, gamma.1, lambda[ n ], x[ n ], y[ n ], lambda.C, lambda.E, mult.effec;

data x in "poisson-x.dat", y in "poisson-y.dat";
inits in "poisson2.in";

{

gamma.0 ~ dnorm( 0.0, 1.0E-4 );  # flat priors for

gamma.1 ~ dnorm( 0.0, 1.0E-4 );  # gamma.0 and gamma.1

for ( i in 1:n ) {

    log( lambda[ i ] ) <- gamma.0 + gamma.1 * x[ i ];
    y[ i ] ~ dpois( lambda[ i ] );
}

lambda.C <- exp( gamma.0 );
lambda.E <- exp( gamma.0 + gamma.1 );
mult.effec <- exp( gamma.1 );
}
```
The multiplicative Poisson model (4) takes longer to run—2,000 burn-in iterations now take about 26 seconds at 333 PC MHz—but still exhibits fairly good mixing, as we’ll see below.
A total of **10,000 iterations** (the chain started essentially in equilibrium, so the burn-in can be absorbed into the monitoring run) reveals that the **multiplicative effect parameter** $e^{\gamma_1}$ in model (4) behaves like an $AR_1$ series with $\hat{\rho}_1 = 0.5$, but the Monte Carlo standard error for the posterior mean is still only about **0.001** with a run of this length.
Additive versus Multiplicative Fit

A burn-in of 2,000 and a monitoring run of 8,000 again yields good MCMC diagnostics and permits a comparison between the additive and multiplicative Poisson models, as in Table 4.4.

Table 4.4. Comparison of inferential conclusions from the additive and multiplicative Poisson models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Posterior Mean</th>
<th>Posterior SD</th>
<th>Central 95% Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>additive</td>
<td>0.943</td>
<td>0.0577</td>
<td>(0.832, 1.06)</td>
</tr>
<tr>
<td>multiplicative</td>
<td>0.945</td>
<td>0.0574</td>
<td>(0.837, 1.06)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Posterior Mean</th>
<th>Posterior SD</th>
<th>Central 95% Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>additive</td>
<td>0.769</td>
<td>0.0521</td>
<td>(0.671, 0.875)</td>
</tr>
<tr>
<td>multiplicative</td>
<td>0.768</td>
<td>0.0518</td>
<td>(0.671, 0.872)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Posterior Mean</th>
<th>Posterior SD</th>
<th>Central 95% Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>additive</td>
<td>-0.174</td>
<td>0.0774</td>
<td>(−0.325, −0.024)</td>
</tr>
<tr>
<td>multiplicative</td>
<td>-0.184</td>
<td>0.0743</td>
<td>(−0.324, −0.033)</td>
</tr>
</tbody>
</table>

With this model it is as if the experimental people’s average underlying rates of hospitalization have been multiplied by 0.82, give or take about 0.07.

The additive and multiplicative effects are similar here, because both are not too far from zero.
Extra-Poisson Variability

However, none of this has verified that the Poisson model is reasonable for these data—the histograms show that the Gaussian model is clearly unreasonable, but the diagnostic plots in WinBUGS and CODA only check on the adequacy of the MCMC sampling, not the model.

In fact we had a good clue that the data are not Poisson back on page 2: the Poisson(\(\lambda\)) distribution has mean \(\lambda\) and also variance \(\lambda\)—in other words, the variance-to-mean-ratio (VTMR) for the Poisson is 1. But

\[
> \text{var( C )} / \text{mean( C )} \\
[1] 1.62599 \\
> \text{var( E )} / \text{mean( E )} \\
[1] 1.322979
\]

i.e., the data exhibit extra-Poisson variability (VTMR > 1).

This actually makes good sense if you think about it, as follows.

The Poisson model assumes that everybody in the control group has the same underlying rate \(\lambda_C\) of hospitalization, and similarly everybody in the experimental group has the same rate \(\lambda_E\).
Unobserved Predictor Variables

In reality it’s far more reasonable to think that each person has his/her own underlying rate of hospitalization that depends on baseline health status, age, and various other things.

Now Hendriksen forgot to measure (or at least to report on) these other variables (he may have hoped that the randomization would balance them between $C$ and $E$)—the only predictor we have is $x$, the experimental status dummy variable—so the best we can do is to lump all of these other unobserved variables together into a kind of “error” term $e$.

This amounts to expanding the second Poisson model (4) above: for $i = 1, \ldots, n = 572$ the new model is

$$(y_i | \lambda_i) \overset{\text{indep}}{\sim} \text{Poisson}(\lambda_i)$$

$$\log(\lambda_i) = \gamma_0 + \gamma_1 x_i + e_i$$  \hspace{1cm} (9)

$$e_i \overset{\text{IID}}{\sim} N(0, \sigma_e^2)$$

$$(\gamma_0, \gamma_1, \sigma_e^2) \sim \text{diffuse.}$$
4.1.3 Random-Effects Poisson Regression

The Gaussian choice for the error distribution is conventional, not dictated by the science of the problem (although if there were a lot of unobserved predictors hidden inside the $e_i$ their weighted sum would be close to normal by the Central Limit Theorem).

Model (9) is an expansion of the earlier model (4) because you can obtain model (4) from (9) by setting $\sigma^2_e = 0$, whereas with (9) we’re letting $\sigma^2_e$ vary and learning about it from the data.

The addition of the random effects $e_i$ to the model is one way to address the extra-Poisson variability: this model would be called a lognormal mixture of Poisson distributions (or a random effects Poisson regression (REPR) model) because it’s as if each person’s $\lambda$ is drawn from a lognormal distribution and then his/her number of hospitalizations $y$ is drawn from a Poisson distribution with his/her $\lambda$, and this mixing process will make the variance of $y$ bigger than its mean.
WinBUGS Implementation

The new WinBUGS model is

\[
\begin{align*}
\text{gamma.0} & \sim \text{dnorm}(0.0, 1.0\times10^{-4}) \\
\text{gamma.1} & \sim \text{dnorm}(0.0, 1.0\times10^{-4}) \\
\text{tau.e} & \sim \text{dgamma}(0.001, 0.001) \\
\text{for} (i \text{ in } 1:n) \{ \\
\quad \text{e}[i] & \sim \text{dnorm}(0.0, \text{tau.e}) \\
\quad \log(\text{lambda}[i]) & \leftarrow \text{gamma.0} + \text{gamma.1} \times x[i] + \text{e}[i] \\
\quad \text{y}[i] & \sim \text{dpois}(\text{lambda}[i]) \\
\} \\
\text{lambda.C} & \leftarrow \exp(\text{gamma.0}) \\
\text{lambda.E} & \leftarrow \exp(\text{gamma.0} + \text{gamma.1}) \\
\text{mult.effect} & \leftarrow \exp(\text{gamma.1}) \\
\text{sigma.e} & \leftarrow 1.0 / \sqrt{\text{tau.e}} \\
\}
\end{align*}
\]

I again use a diffuse $\Gamma(\epsilon, \epsilon)$ prior (with $\epsilon = 0.001$) for the precision $\tau_\epsilon$ of the random effects.
With a model like that in equation (9), there are \( n \) random effects \( e_i \) that need to be sampled as nodes in the graph along with the fixed effects \((\gamma_0, \gamma_1)\) and the variance parameter \( \sigma^2_e \), which—in earlier releases of the software, at least—made it more crucial to give WinBUGS good starting values.

Here WinBUGS release 1.3 has figured out that random draws like \( 1.66 \cdot 10^{-316} \) result from the generic (and quite poor) initial values \((\gamma_0, \gamma_1, \tau_e) = (0.0, 0.0, 1.0)\) and has refused to continue sampling.
Warning WinBUGS can fail, particularly in random-effects models, when you give it initial values that are not very close to the final posterior means; an example in release 1.3 is the REPR model (9) on the IHGA data with the “generic” starting values \((\gamma_0, \gamma_1, \tau_e) = (0.0, 0.0, 1.0)\).

When this problem arises there are two ways out in WinBUGS: trial and error, or a calculation (see below).

NB MLwiN does not have this problem—it gets its starting values from maximum likelihood (the mode of the likelihood function is often a decent approximation to the mean or mode of the posterior).

Technical note. To get a decent starting value for \(\tau_e\) in model (9) you can calculate as follows: renaming the random effects \(\eta_i\) to avoid confusion with the number \(e\),

\[
\begin{align*}
(1) \quad V(y_i) &= V[E(y_i | \eta_i)] + E[V(y_i | \eta_i)], \text{ where} \\
(2) \quad (y_i | \eta_i) &\sim \text{Poisson}(e^{\gamma_0 + \gamma_1 x_i + \eta_i}), \text{ so} \\
E(y_i | \eta_i) &= V(y_i | \eta_i) = e^{\gamma_0 + \gamma_1 x_i + \eta_i}. \text{ Then (3)} \\
V[E(y_i | \eta_i)] &= V(e^{\gamma_0 + \gamma_1 x_i + \eta_i}) = e^{2(\gamma_0 + \gamma_1 x_i)}V(e^{\eta_i}) \text{ and} \\
E[V(y_i | \eta_i)] &= E(e^{\gamma_0 + \gamma_1 x_i + \eta_i}) = e^{\gamma_0 + \gamma_1 x_i}E(e^{\eta_i}). \text{ Now (4) } e^{\eta_i} \text{ is lognormal with mean 0 and variance } \sigma_e^2 \text{ on the log scale, so} \\
E(e^{\eta_i}) &= e^{\frac{1}{2}\sigma_e^2} \text{ and } V(e^{\eta_i}) = e^{\sigma_e^2}(e^{\sigma_e^2} - 1), \text{ yielding finally} \\
V(y_i) &= e^{2(\gamma_0 + \gamma_1 x_i) + \frac{1}{2}\sigma_e^2} + e^{\gamma_0 + \gamma_1 x_i + \sigma_e^2}(e^{\sigma_e^2} - 1). \text{ (5) Plugging in } x_i = 0 \text{ for the } C \text{ group, whose sample variance is 1.54, and using the value } \gamma_0 = -0.29 \text{ from runs with previous models, gives an equation for } \sigma_e^2 \text{ that can be solved numerically, yielding } \sigma_e^2 \approx 0.5 \text{ and } \tau_e \approx 2.
\end{align*}
\]
Interestingly, WinBUGS release 1.4 is able to sample successfully with the generic starting values \((\gamma_0, \gamma_1, \tau_e) = (0.0, 0.0, 1.0)\), although of course a longer burn-in period would be needed when they’re used; you have to try truly absurd initial values to get it to fall over, and when it does so the error message ("Rejection1") in the lower left corner is more discreet.
With a **better set of initial values**—\((\gamma_0, \gamma_1, \tau_e) = (-0.058, -0.21, 2.0)\), obtained from (a) the earlier Poisson models (in the case of the regression parameters \(\gamma_i\)) and (b) either a calculation like the one on the bottom of page 20 or trial and error—WinBUGS is able to make progress, although this model takes a **fairly long time to fit** in release 1.4: a burn-in of 1,000 takes 33 seconds at 333 PC MHz (the code runs about twice as fast in release 1.3 for some reason).

A monitoring run of **5,000** iterations reveals that the random effects make everything **mix more slowly**: \(\lambda_C\) (this page) and \(\lambda_E\) and the multiplicative effect (next page) all behave like \(AR_1\) series with \(\hat{\rho}_1 \doteq 0.7, 0.5,\) and \(0.6,\) respectively.
Learning about $\sigma_e$ in this model is slow: its autocorrelation function is that of an $AR_1$ with a high value of $\hat{\rho}_1$ (equation (55) on page 76 of part 3 of the lecture notes gives $\hat{\rho}_1 \doteq 0.92$).

The MCSE of the posterior mean for $\sigma_e$ based on 5,000 draws is $0.005182$—to get this down to (say) $0.001$ I need to increase the length of the monitoring run by a factor of $(0.005182)^2 \doteq 26.9$, meaning a total run of about $(26.9)(5,000) \doteq 134,000$ iterations (this takes about half an hour at 1 PC GHz).