Expectation of mean squares

Expected error mean square:

$$
\begin{aligned}
\mathrm{E}[MS_{Err.}] &= \mathrm{E}\left[\frac{SS_{Err.}}{N-I}\right] = \frac{1}{N-I}\mathrm{E}\left[\sum_{i=1}^{I}\sum_{j=1}^{J}(y_{ij}-\bar{y}_i)^2\right] \\
&= \frac{1}{N-I}\mathrm{E}\left[\sum_{i=1}^{I}\sum_{j=1}^{J}(y_{ij}^2 - 2y_{ij}\bar{y}_i + \bar{y}_i^2)\right] \\
&= \frac{1}{N-I}\mathrm{E}\left[\sum_{i=1}^{I}\sum_{j=1}^{J}y_{ij}^2 - 2J\sum_{i=1}^{I}\bar{y}_i^2 + J\sum_{i=1}^{I}\bar{y}_i^2\right] \\
&= \frac{1}{N-I}\mathrm{E}\left[\sum_{i=1}^{I}\sum_{j=1}^{J}y_{ij}^2 - J\sum_{i=1}^{I}\bar{y}_i^2\right] \\
&= \frac{1}{N-I}\mathrm{E}\left[\sum_{i=1}^{I}\sum_{j=1}^{J}y_{ij}^2 - \frac{1}{J}\sum_{i=1}^{I}(\sum_{j=1}^{J}\bar{y}_{ij})^2\right] \\
&= \frac{1}{N-I}\mathrm{E}\left[\sum_{i=1}^{I}\sum_{j=1}^{J}(\mu+\alpha_i+e_{ij})^2 - \frac{1}{J}\sum_{i=1}^{I}(\sum_{j=1}^{J}(\mu+\alpha_i+e_{ij}))^2\right] \\
&= \frac{1}{N-I}\left[N\mu^2 + J\sum_{i=1}^{I}\alpha_i^2 + N\sigma^2 - N\mu^2 - J\sum_{i=1}^{I}\alpha_i^2 - I\sigma^2\right] \\
&= \left(\frac{N-I}{N-I}\right)\sigma^2 = \sigma^2
\end{aligned}
$$

Expected treatment mean square (balanced designs):

$$\begin{aligned}
\mathrm{E}[MS_{Trt.}] = \mathrm{E}\left[\frac{SS_{Trt.}}{I-1}\right] &= \frac{1}{I-1}\mathrm{E}\left[J\sum_{i=1}^{I}(\bar{y}_i - \bar{y}_{..})^2\right] \\
&= \frac{1}{I-1}\mathrm{E}\left[J\sum_{i=1}^{I}(\bar{y}_i^2 - 2\bar{y}_i\bar{y}_{..} + \bar{y}_{..}^2)\right] \\
&= \frac{1}{I-1}\mathrm{E}\left[J\sum_{i=1}^{I}\bar{y}_i^2 - 2J\bar{y}_{..}\sum_{i=1}^{I}\bar{y}_i + JI\bar{y}_{..}^2\right] \\
&= \frac{1}{I-1}\mathrm{E}\left[J\sum_{i=1}^{I}\bar{y}_i^2 - 2JI\bar{y}_{..}^2 + JI\bar{y}_{..}^2\right] \\
&= \frac{1}{I-1}\mathrm{E}\left[J\sum_{i=1}^{I}\bar{y}_i^2 - JI\bar{y}_{..}^2\right] \\
&= \frac{1}{I-1}\mathrm{E}\left[J\sum_{i=1}^{I}(\frac{1}{J}\sum_{j=1}^{J}y_{ij})^2 - JI(\frac{1}{JI}\sum_{i=1}^{I}\sum_{j=1}^{J}y_{ij})^2\right] \\
&= \frac{1}{I-1}\mathrm{E}\left[\frac{1}{J}\sum_{i=1}^{I}(\sum_{j=1}^{J}y_{ij})^2 - \frac{1}{JI}(\sum_{i=1}^{I}\sum_{j=1}^{J}y_{ij})^2\right] \\
&= \frac{1}{I-1}\mathrm{E}\left[\frac{1}{J}\sum_{i=1}^{I}(\mu + \alpha_i + e_{ij})^2 - \frac{1}{JI}(\sum_{i=1}^{I}\sum_{j=1}^{J}(\mu + \alpha_i + e_{ij}))^2\right] \\
&= \frac{1}{I-1}\left[N\mu^2 + J\sum_{i=1}^{I}\alpha_i^2 + I\sigma^2 - \frac{(JI\mu)^2}{JI} - \frac{JI\sigma^2}{JI}\right] \\
&= \frac{1}{I-1}\left[(I-1)\sigma^2 + J\sum_{i=1}^{I}\alpha_i^2\right] = \sigma^2 + \frac{J\sigma_{i=1}^{I}\alpha_i^2}{I-1}
\end{aligned}$$

Benjamini-Hochberg (BH)

The disadvantage of using correction methods that control for the family-wise error rate (i.e. Bonferroni or Sidak) is that these methods don't scale well with the number of tests performed. For instance, in many 'omics-type analyses we performing pair-wise comparisons for thousands of response variables (or for GWAS many, many predictors). These approaches control for FWER at the expense of statistical power. The BH procedure seeks to control the false discovery rate (FDR) – the expected proportion of tests that are falsely rejected. The benefit is that it scales much better with the number of tests, allowing true positives to still be recovered. The method is simple and is based on a ranking of p-values. Briefly, we perform some number of tests (m), rank the p-values from small to large, and adjust each p-value according to its rank (i; from smallest to largest) via $p_i \frac{m}{i}$. It is computed from the largest p-value to the smallest p-value. R produces "adjusted" p-values, which was not mentioned in the original paper (Benjamini and Hochberg, 1995). The adjustment method is described in Benjamini, Heller and Yekutieli (2009) and is given by

$$p_i^{adj.} = \min\left\{\min_{j \geq i}\left\{\frac{mp_j}{j}\right\}, 1\right\}$$

With the ranked p-values, this basically says that for a given p-value (i), if a smaller adjusted p-value exists (j) then use that p-value.

We can implement this approach in R using the code below. Suppose we perform ten tests and recover the following p-values

```
> p = c(0.4510, 0.2000, 0.1750, 0.1500, 0.1000, 0.0500, 0.0250,
        0.0020, 0.0015, 0.0010)
> ranks = 10:1
> m = 10
> new_ps = cummin(m/ranks * p)
> round(new_ps, 3)
 [1] 0.451 0.222 0.219 0.214 0.167 0.100 0.062 0.007 0.007 0.007
> round(p.adjust(p, method = "BH"), 3)
 [1] 0.451 0.222 0.219 0.214 0.167 0.100 0.062 0.007 0.007 0.007
```