Class 2: Likelihood and Bayesian Imputation Methods

Andrew Parnell
andrew.parnell@mu.ie

Maynooth University
National University of Ireland Maynooth

https://andrewcparnell.github.io/mda_course

PRESS RECORD

In this class . . .

- Revision of likelihood and Bayes
- Imputation via Bayesian inference
- Approximate methods for imputation
- The `mice` package/algorithm

Revision of likelihood

- If we are in a complete data situation we need to compute $P(Y|\theta)$ where Y are our observations, and θ is the parameter(s) of our model. This is the **likelihood**
- If we assume that the data are (conditionally) independent we have $P(Y|\theta) = \prod_{i=1}^{n} P(Y_i|\theta)$
- In linear regression we split the data up into a response Y and covariates X, and θ would now be our intercept, our slope, and our residual standard deviation
- To get this to work we need to choose a probability distribution for P. For linear regression this is a normal distribution

Calculating the likelihood

For the Whiteside data

```
library(MASS)
y = whiteside$Gas
x = whiteside$Temp
prod(dnorm(y, 5.5 - 0.3*x, 1))
```

```
## [1] 8.780779e-32
```

This is pretty small so most people work with the log likelihood to keep numerical stability

```
sum(dnorm(y, mean = 5.5 - 0.3*x, sd = 1, log = TRUE))
```

```
## [1] -71.51016
```

Maximising the likelihood

- One way to find the 'best' parameters is to try lots and lots of different values and take the ones that provide the biggest log likelihood. This is very inefficient
- Another way is to use mathematics - we can often maximise the likelihood using calculus. (We are not going to do this)
- R has a number of very efficient built-in optimisation routines (e.g. `nlminb` and `optimx`) which will find the best values for us

Likelihood inference for missing data

- Now suppose we have a missing observation:

```
y[5] = NA
```

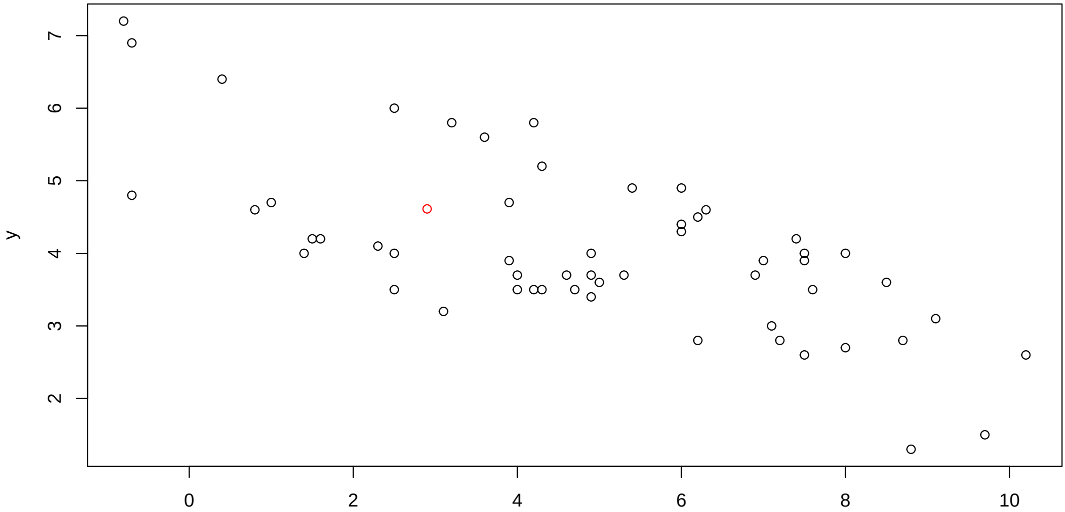
- Our parameters are now $\theta = (y_5, \alpha, \beta, \sigma)$. In R we could optimise via:

```
nll = function(theta) {
  newy = y; newy[5] = theta[1]
  -sum(dnorm(newy, mean = theta[2] + theta[3]*x, sd = theta[4], log = TRUE
}
answer = nlminb(rep(2, 4), nll, lower = c(-Inf, -Inf, -Inf, 0))
print(answer$par)
```

```
## [1]  4.6122369  5.4374802 -0.2845665  0.8304745
```

Did it work?

```
plot(x, y)
points(x[5], answer$par[1], col = 'red')
```



Bayes' theorem

- Treating the missing data as parameters and maximising the likelihood works fine for large data sets with low degrees of missingness
- To get uncertainties we rely on large sample approximations which rely on asymptotic normality. This sometimes leads to inappropriate confidence intervals
- Frequentist inference also often relies on p-values which are easily gamed.
- There is a saviour to the rescue...

Enter Bayes

An essay towards solving a problem on the doctrine of chances (1763)

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

How does Bayesian statistics work?

- Bayesian statistics is based on an interpretation of Bayes' theorem
- All quantities are divided up into *data* (i.e. things which have been observed) and *parameters* (i.e. things which haven't been observed)
- We use Bayes' interpretation of the theorem to get the *posterior probability distribution*, the probability of the unobserved given the observed
- Bayes' equation is usually written mathematically as:

$$p(\theta|x) \propto p(x|\theta) \times p(\theta)$$

This reads as posterior is proportional to prior times likelihood

Choosing a prior

There are several choices when it comes to specifying prior distributions:

- *Informative*, when there is information from a previous study, or other good external source, e.g $\theta \sim N(-30, 1.5^2)$
- *Vague*, when there is only weak information, perhaps as to the likely range of the parameter e.g. $\theta \sim N(0, 10^2)$
- *Flat*, when there is no information at all about a parameter (very rare).

In most cases, priors can only be understood in the context of the data, for example a $N(0, 100)$ prior might be uninformative for one data set but extremely informative for another

Fitting a Bayesian model

- The first step is to choose the prior and likelihood probability distributions
- Then we fit the model to obtain a posterior distribution
- These posterior distributions appear as *samples* from the posterior probability distribution rather than direct estimates of means/standard deviations or equations
- We can thus obtain complicated probability distributions for our parameters which do not need to be asymptotically normal

Gibbs sampling

- Fitting a complicated model to get a posterior distribution is hard, and is not usually possible using maximisation techniques
- A useful trick is to use a Gibbs sampler where we sample each parameter in turn conditional on the others
- If we have set of parameters $\theta = (\theta_1, \ldots, \theta_k)$, then we create:

$$P(\theta_j|\theta^{(-j)}) \propto P(Y|\theta)P(\theta_j)$$

- Quite often the probability distributions simplify depending on the priors and the likelihoods
- Even when they simplification isn't possible there are other techniques to simulate from these probability distributions
- The algorithm works with starting guesses for θ, then iterates through all the parameters over and over again and, theoretically, is guaranteed to end up at the posterior distribution

Fitting a Bayesian linear regression model in JAGS

```
model_code ='
model {
  # Likelihood
  for(i in 1:N) {
    y[i] ~ dnorm(intercept + slope*x[i], residual_sd^-2)
  }
  # Priors
  intercept ~ dnorm(0,10^-2)
  slope ~ dnorm(0,10^-2)
  residual_sd ~ dunif(0,10)
}
'
```

Running the model

```
library(R2jags)
mod_par =  c("intercept",
             "slope",
             "residual_sd")
model_run = jags(data =
                   list(N = length(y),
                        y = y, x = x),
                 parameters.to.save =
                   mod_par,
                 model.file =
                   textConnection(model_code))
```

Output

```r
print(model_run)
```

```
## Inference for Bugs model at "4", fit using jags,
##  3 chains, each with 2000 iterations (first 1000 discarded)
##  n.sims = 3000 iterations saved
##             mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat n.ef
## intercept     5.442   0.242   4.975   5.283   5.445   5.604   5.910 1.001  300
## residual_sd   0.875   0.089   0.722   0.810   0.867   0.930   1.070 1.001  300
## slope        -0.286   0.044  -0.372  -0.315  -0.286  -0.256  -0.199 1.001  200
## deviance    139.796   2.556 136.923 137.921 139.146 140.913 146.430 1.001  300
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 3.3 and DIC = 143.1
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Bayesian computation with JAGS and Stan

- JAGS (Just Another Gibbs Sampler) fits Bayesian models using Gibbs Sampling to update the parameters in turn
- It has its own language and so the code needs to be stored in a text string or a separate file
- It's usually pretty fast and can run models with many parameters
- A related language, Stan, is slightly more fashionable but a bit slower to run (at first) and harder to teach! (But also worth looking at for missing data analysis)
- We will do some missing data analysis with JAGS in the computation session

An example of JAGS code for missing data

```
model_code = '
model
{
  # Likelihood
  for (i in 1:n) {
    y[i] ~ dnorm(intercept + slope * x[i], residual_sd^-2)
  }
  # Priors for the missing x values
  for(k in 1:n_miss_x) {
    x[miss[k]] ~ dunif(min_x, max_x)
  }
  # Priors
  intercept ~ dnorm(0, 100^-2)
  slope ~ dnorm(0, 100^-2)
  residual_sd ~ dunif(0, 100)
}
'
```

Why doesn't everybody use Bayesian inference for MDA?

- This approach works well for small amounts of missing data, and/or when there is good prior information on what those missing values are
- It doesn't work so well for larger missing data sets as the parameter gets too big and the models get very slow or even break
- Instead what people tend to do is use an approximation; use a Bayesian model to impute the missing values, then fit the model using these imputed values as though they were the prior
- This is a bit of a cheat, but it does seem to work well in simulation studies

Imputing first, model later

- If we take the more practical route of imputing first and modelling later, we can still use the tools of Bayesian inference
- We now treat our observed and missing data as though it was a multivariate probability distribution and try to learn the parameters
- If all the data are continuous, a common distribution to choose is the multivariate normal distribution
$$Y_i \sim N(\mu, \Sigma)$$
where Y_i is the vector of observations, μ a vector of means, and Σ the covariance matrix
- With missing data this can be a fiddly likelihood to write down

Multivariate imputation with ignorable monotone missingness

- Recall that missingness mechanism is ignorable if we don't need to include a model for the M missingness matrix
- Also recall that monotone missing means that the variables $Y_{i1}, \ldots, Y_{ip}$ can be ordered such that missingness occurs in a staircase pattern; once a variable is missing on variable j it is also missing for variables $j + 1, \ldots, p$
- Let's look at an example

Ignorable monotone missingness example

- Here's a 2D example (which will always satisfy monotone missingness):

```
whiteside3 = whiteside[,2:3]
whiteside3[35:56,2] = NA # Last 22 values on Gas are missing
```

- The job now is just to predict those missing values. We need a likelihood $P(Y_{\text{obs}}|Y_{\text{mis}}, \mu, \Sigma)$ but this is hard to write down, e.g.

```
library(mvtnorm)
sum(dmvnorm(whiteside3, mean  = rep(5,2), sigma = diag(2), log = TRUE))
[1] NA
```

- But we can use our trick from earlier! Let Y_1 and Y_2 be the two columns:

$$P(Y_1, Y_2|\mu, \Sigma) = P(Y_2|Y_1, \mu, \Sigma) \times P(Y_1|\mu, \Sigma)$$

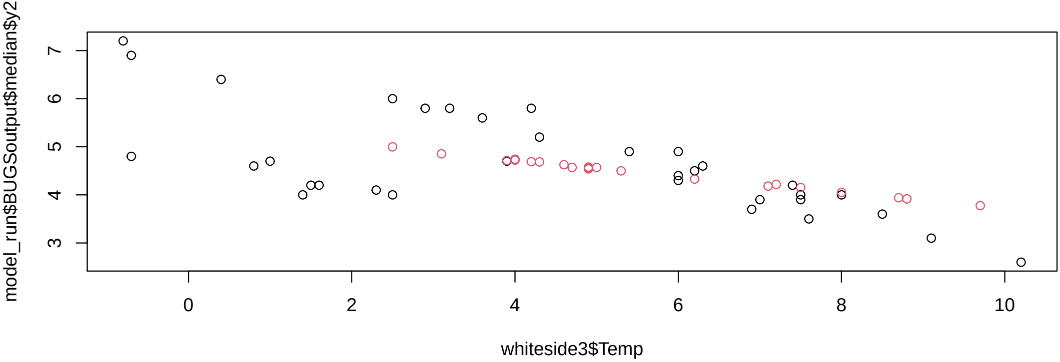
  The first term we can write using our conditioning formula (see last class), and the second has no missing values!

JAGS code for the ignorable monotone missingness case

```
model_code = '
model
{
  # Likelihood
  for (i in 1:N) {
    y1[i] ~ dnorm(mu1, sd1^-2)
    y2[i] ~ dnorm(mu2 + (sd2/sd1)*rho*(y1[i] - mu1), ((1-rho^2)*(sd1^2))^-2)
  }
  # Priors
  mu1 ~ dnorm(0, 100^-2)
  mu2 ~ dnorm(0, 100^-2)
  sd1 ~ dunif(0, 100)
  sd2 ~ dunif(0, 100)
  rho ~ dunif(-1, 1)
}
'
```

Running the model

```
model_run = jags(data = list(N = nrow(whiteside3),
                              y1 = whiteside3$Temp, y2 = whiteside3$Gas),
                 parameters.to.save = c("y2"),
                 model.file = textConnection(model_code))
plot(whiteside3$Temp, model_run$BUGSoutput$median$y2,
     col = is.na(whiteside3$Gas)+1)
```



An alternative to Gibbs: the EM algorithm

- Gibbs sampling can be a little bit slow if there are lots of missing values
- A neat alternative is the **Expectation Maximisation** (EM) algorithm which finds the posterior mode in the parameters
- This means that it ignore the parameter uncertainty in the imputation but will give us best guesses of the parameters in a model (in the previous case μ and Σ)
- The method works by guessing at the missing values, then finding the ML estimates of the now 'complete data', before re-guessing the missing values again

Mathematics of EM

For a missing data analysis, EM works by first finding the expected complete-data likelihood for some parameter guesses $\theta^{(t)}$ at iteration t:

- The E step

$$Q(\theta|\theta^{(t)}) = \int P(Y_{\text{obs}}, Y_{\text{mis}}|\theta^{(t)}) \times P(Y_{\text{mis}}|Y_{\text{obs}}, \theta^{(t)}) dY_{\text{mis}}$$

- The M step then maximises this quantity:

$$\theta^{(t+1)} = \arg\max_{\theta} Q(\theta|\theta^{(t)})$$

It's easier to explain with an example!

EM example for missing data

Let's fit the monotone missingness example from above. If the first r values of y_2 are known then our missing parameters are

$$\mu_1, \mu_2, \sigma_{11}, \sigma_{12}, \sigma_{22}, y_{2,r+1}, \ldots, y_{2,n}$$

Some very dull algebra gives:

$$y_{2i}^{(k)} = \mu_2^{(k)} + \frac{\sigma_{12}^{(k)}}{\sigma_{11}^{(k)}} \left(y_{1i} - \mu_1^{(k)} \right)$$

$$\sigma_{jj}^{(k+1)} = \left(\sum (y_{ji}^{(k)})^2 - \frac{\left(\sum y_{ji}^{(k)} \right)^2}{n} \right) / n$$

$$\sigma_{12}^{(k+1)} = \left(\sum y_{1i} y_{2i}^{(k)} - \frac{\left(\sum y_{1i} \sum y_{2i}^{(k)} \right)^2}{n} \right) / n, \;\; \mu_2^{(k)} = \frac{\sum y_{2i}^{(k)}}{n}$$

Some comments on the EM algorithm

- The EM algorithm often works in general missingness problems and not just in ignorable and/or MAR approaches
- However it does require a bit of mathematics to set up and can be slow to converge
- The reason its helpful is because you only have to find the expected value of the likelihood over the missing data, and not completely integrate it out
- It can be viewed as a special 'optimised' version of Gibbs sampling, or as a coordinate ascent algorithm
- There are dozens of extensions to EM, some of which will be appropriate for more advanced problems

Summary so far

- There are two broad approaches, either:
  1. Estimate your missing data at the same time as fitting your model, or
  2. Fill in your missing values using multiple imputation and then fit your model using those values
- We can do either for ignorable data using Bayesian methods and/or the EM algorithm
- These however can be slow to fit and require a bit of heavy maths to set up
- None of the methods we have looked at so far works for mixed continuous/categorical data
- `mice` simplifies all this by just setting up models for option (2) without worrying too much about the theory

The `mice` approach

- `mice` is an R package that has been around for at least 15 years
- It uses a method called *Fully Conditional Specification* (FCS) in which the conditional distributions:
$$P(Y_j | Y^{(-j)}, X, M, \theta)$$
are all specified up front
- It looks a bit like a Gibbs sampling algorithm but is not guaranteed to produce a valid posterior distribution

The mice algorithm

1. Specify imputation models for $Y_j | Y_{(-j)}, X, M, \theta$ with $j = 1, \ldots, p$
2. For each variable guess at starting values of the imputed data $\hat{Y}_j^{t=0}$
3. Loop over iterations t and variables j. Define $\dot{Y}_{-j}^t$ as the current 'complete' data set
4. Draw parameters $\dot{\theta}_j^t \sim P(\theta_j^t | Y_j^{\text{obs}}, \dot{Y}_{-j}^t, M)$
5. Draw imputations $\dot{Y}_j^t \sim P(Y_j^{\text{mis}} | Y_j^{\text{obs}}, \dot{Y}_{-j}^t, M, \dot{\theta}_j^t)$

Key aspects of the `mice` approach

- It allows for great flexibility because if the data are different types (continuous, counts, factors, etc) they can all have their own imputation model
- It usually only produces 5-40 imputed data values. Compare this with the Bayesian JAGS approach which produces many thousands
- In certain circumstances (e.g. if all the conditional distributions are normal) it will be equivalent to the joint modelling approach
- Because of its set up, the issue of monotone missingness is mostly irrelevant (though it might make the algorithm work better because there is an easier to find valid joint distribution)

Modelling with imputed data

The usual steps to working with `mice` is:

1. Impute your data multiple times (usually `m=5` or more)
2. Fit your model to each imputed data set
3. Pool your results to get an overview of the fitting approach whilst marginalising over the missing data

- `mice` uses special S3 classes and functions for each of these different steps to make the code clearer

Pooling different models

- The pooling step in `mice` is one of the weaker aspects of the package
- Because the model is not set in a fully Bayesian framework we have to rely on traditional asymptotic normality of the parameters
- If the parameters are constrained then an appropriate function (e.g. log for positive data, logit for (0,1) data) must be used to 'unconstrain' it
- You can then perform weighted averaging, and some slightly obscure mathematics to combine the test statistics to produce 'valid' p-values

Sensitivity checking

- Some get confused because `mice` produces a number of imputed data sets (argument `m`) over a set of iterations (argument `maxit`)
- `m` is the number of imputed data sets
- `maxit`, is the number of iterations for *each* imputed variable and data set
- We usually plot the iterations for each variable (and imputation) to see if they look stable. Usually we only need a few iterations in these algorithms
- If some variables are known to be related (e.g. through prior information) this can be specified with the argument `predictorMatrix`

Summary

- We have seen the gold standard for imputation through Bayesian inference, though this doesn't always work well in situations where there is a high degree of missingness
- We have seen an alternative technique which first imputes the data and then combines the parameter estimates
- This can be done either using Bayes again, or through EM, or via FCS with `mice`
- Now on to actually use the package!