Measurement Error in Regression

Measurement error is a huge topic in epidemiology, because in our field it is *extremely common* to have variables that are not measured properly. For example, many human measurements fluctuate over time (blood pressure, cholesterol levels, heart rate), and others may come from questionnaires not necessarily answered honestly (income, daily food intakes, weight). Other errors may occur because of machine errors (bone mineral density, voting machines, radio carbon dating), and recall errors (past smoking history, work exposure history, living near high power lines).

When seeing this list, you realize that just about every linear or logistic model run in an epidemiology study will have one or more mismeasured variables. In some studies, it is possible that just about all variables have some degree of measurement error.

Given the ubiquity of mismeasured variables, one must consider the effects on estimation of regression coefficients.

First, let's consider what design and other factors influence the success of a regression. Following that, we will see some examples of errors in estimation that arise from mismeasured variables, and some methods (both frequentist and Bayesian) for adjusting for measurement error.

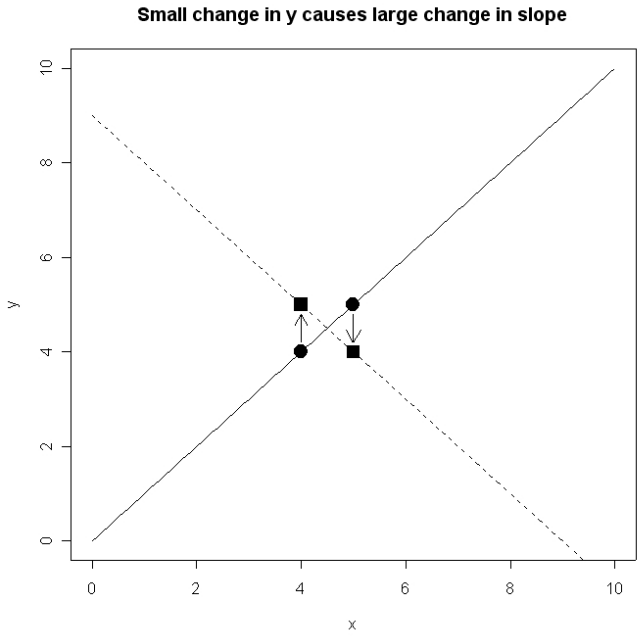
Design Issues And Effects Of Measurement Error

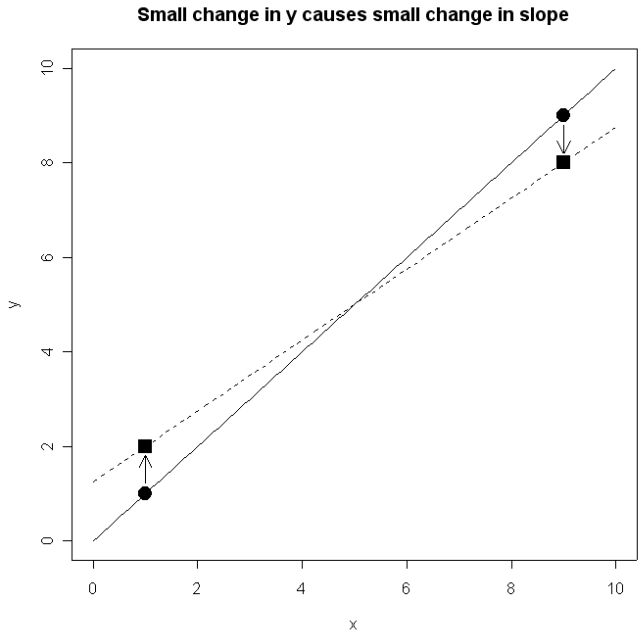
Suppose we are designing an experiment that will be analyzed as a linear regression between two variables, say x and y. We will choose values of x to observe (so we "have control" over values of x), and for each value of x we choose, we will observe the value of y that is associated with it.

Question: What values of x should we choose in order the "most accurately" estimate the linear regression line?

Points to Consider:

1. Values of x that are further apart from each other tend to produce more stable estimates. Consider the following graphs:



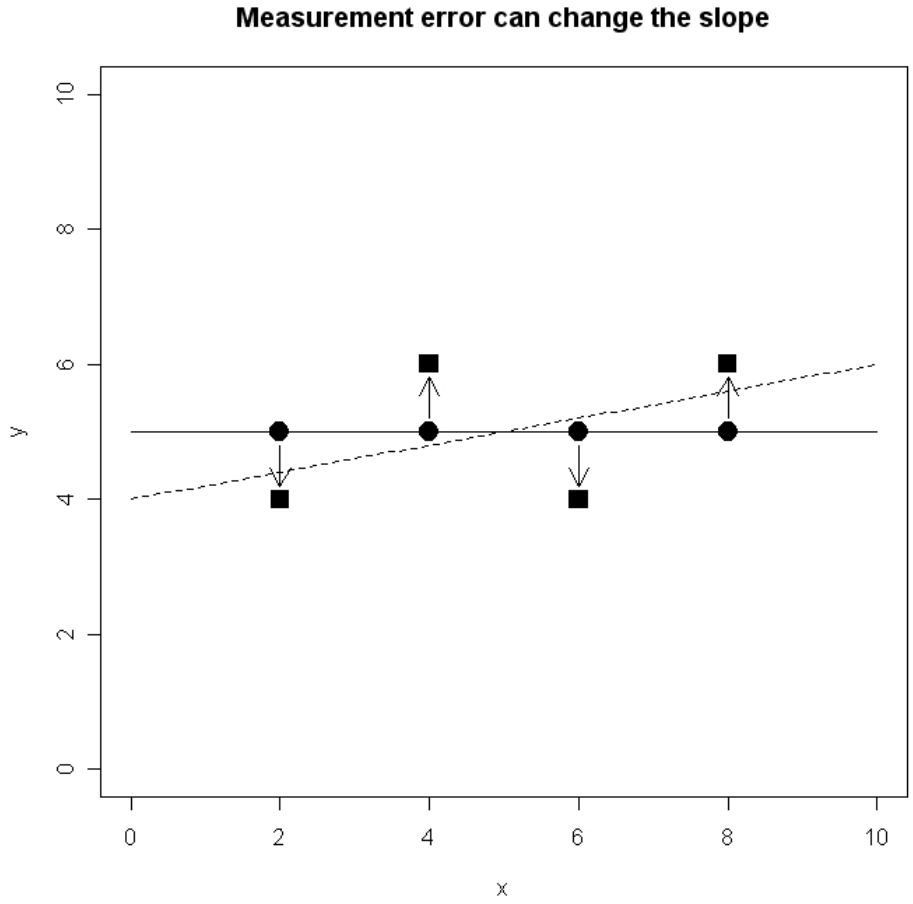


2. As always, a larger sample size leads to more accurate estimation, since the α and β coefficients will be estimated more accurately.

3. If there is a chance that the relationship is not exactly linear (almost always the case), then selecting x values spread out along the feasible range will allow you to explore linearity.

4. Consider taking some repeated measurements at some x values, to allow for investigation of whether σ^2 changes with x.

5. Try to measure x and y as accurately as possible, since **measurement error** can severely compromise your attempts to accurately determine the relationship between x and y. Consider the graph below:



Point 5 above illustrates the potential effects of measurement error. Let's consider a simple example:

Suppose a true relationship between x and y is linear, with

$$y = \alpha + \beta * x$$

One of the following four situations necessarily exists:

1. Both x and y are perfectly measured, so all estimates are unbiased.
2. The dependent variable y is measured with error, but x is measured without error. This causes no bias in the estimation of β, but the residual standard error is increased, causing all parameters to be estimated less precisely.
3. The dependent variable y is measured without error, but x is measured with error. This causes bias in the estimation of β. The bias is towards 0, i.e., measurement error makes x seem less important than it really is.

4. Both x and y are measured with error. This causes both bias towards zero when estimating β, and less accurate estimation of all parameters.

The effect on precision arising from measurement error in y is easy to grasp. If the measurement error is random about 0, then the measurement error variance is added to the usual residual variance. The net effect is a larger residual variance.

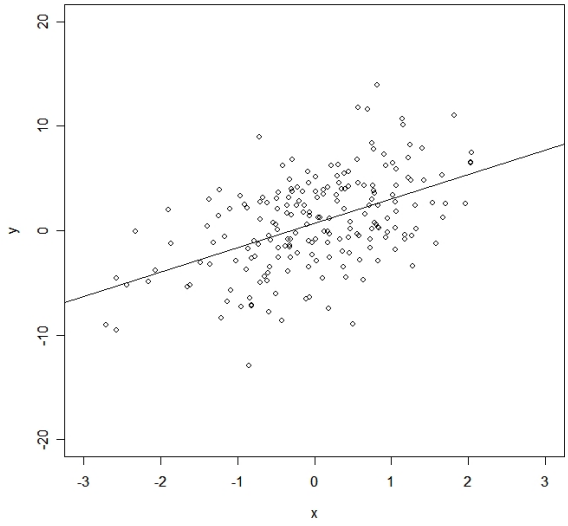
Let's illustrate with an example, and then see how to adjust for measurement error, at least in x.

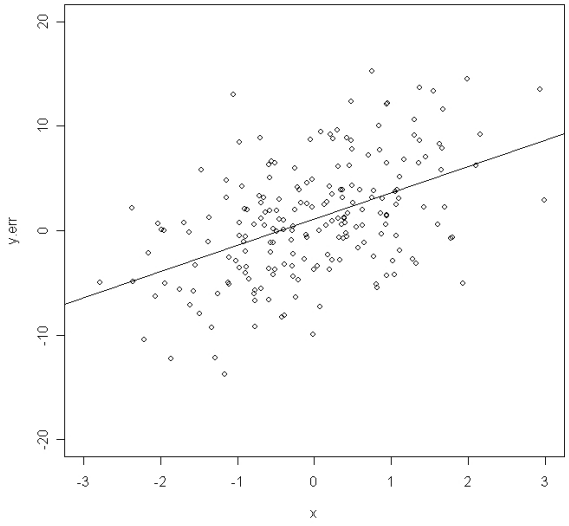
```
#   Suppose that in fact y = 1 + 2x

#  1. First simulate without measurement error
#        (but with usual residual error)

> x <- rnorm(200)

> y <- 1 + 2 * x + rnorm(200, mean=0, sd=4)

> output <- lm(y~x)

> summary(output)

Call:
lm(formula = y ~ x)

Residuals:
      Min        1Q    Median        3Q       Max
-11.62529  -2.54214   0.07242   2.97602  11.32980

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.7118     0.2780   2.560   0.0112 *
x             2.3408     0.2933   7.981 1.15e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.931 on 198 degrees of freedom
Multiple R-Squared: 0.2434,     Adjusted R-squared: 0.2396
F-statistic: 63.69 on 1 and 198 DF,  p-value: 1.155e-13


#  Everything works perfectly.
```

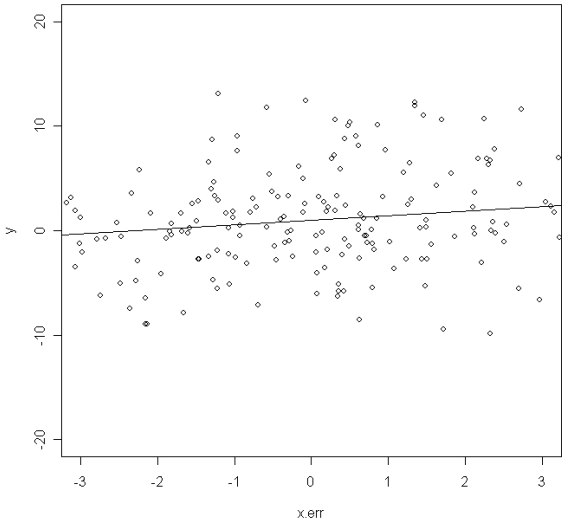
```
#  Now for situation 2, y has error, x has none
#  Take original y, and add some error to it.

> y.err <- y + rnorm(200, mean=0, sd=2)

> output <- lm(y.err~x)

> summary(output)

Call:
lm(formula = y.err ~ x)

Residuals:
     Min       1Q   Median       3Q      Max
-11.7856  -2.6914   0.1838   2.4809  14.9860

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.6623     0.3025   2.189   0.0297 *
x             2.3173     0.3192   7.260 8.63e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.277 on 198 degrees of freedom
Multiple R-Squared: 0.2102,     Adjusted R-squared: 0.2063
F-statistic: 52.71 on 1 and 198 DF,  p-value: 8.626e-12

#  No bias, but note that standard errors have increased
#  by about 10%.

#  In fact, new residual variance = old variance + error variance
#                 4.277^2 = 18.3  = 3.931^2 + 2^2 = 15.4 + 4 = 19.4

#  3.  Now try with error in x but not on y

> x.err <- x + rnorm(200, mean=0, sd=2)

> output <- lm(y~x.err)

> summary(output)

Call:
lm(formula = y ~ x.err)

Residuals:
```

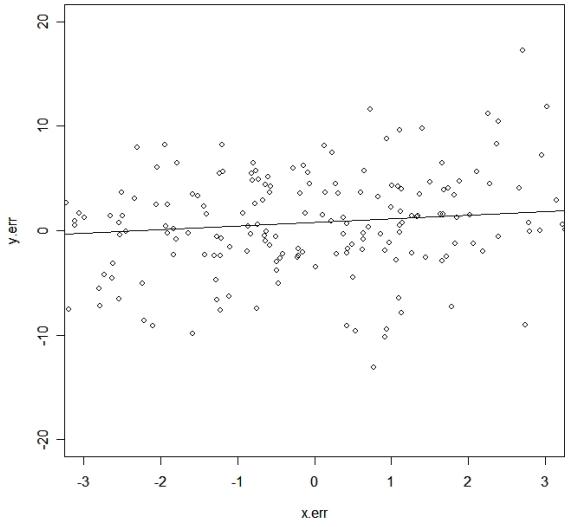
```
     Min       1Q   Median       3Q      Max
-14.1125  -2.8802   0.3518   2.7018  11.6179

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.8346     0.3108   2.686 0.007854 **
x.err         0.4965     0.1419   3.499 0.000576 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.385 on 198 degrees of freedom
Multiple R-Squared: 0.05825,    Adjusted R-squared: 0.05349
F-statistic: 12.25 on 1 and 198 DF,  p-value: 0.0005759

#  Note the very large bias in beta towards 0!

#  4.  Now error in both x and y

> output <- lm(y.err~x.err)

> summary(output)

Call:
lm(formula = y.err ~ x.err)

Residuals:
      Min        1Q    Median        3Q       Max
-14.11596  -2.75692   0.07359   3.05007  15.55696

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.7631     0.3368   2.266   0.0245 *
x.err         0.3470     0.1538   2.257   0.0251 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.753 on 198 degrees of freedom
Multiple R-Squared: 0.02507,    Adjusted R-squared: 0.02015
F-statistic: 5.092 on 1 and 198 DF,  p-value: 0.02513

#  Note large bias towards zero, and increased residual error.

#  Can look at plots of these four situations
#  (scatter plot + best fitting line)
```

```
plot(x,y, xlim=c(-3,3), ylim=c(-20,20))
abline(lsfit(x,y)$coefficient[1], lsfit(x,y)$coefficient[2])

plot(x, y.err, xlim=c(-3,3), ylim=c(-20,20))
abline(lsfit(x,y.err)$coefficient[1], lsfit(x,y.err)$coefficient[2])

plot(x.err, y, xlim=c(-3,3), ylim=c(-20,20))
abline(lsfit(x.err,y)$coefficient[1], lsfit(x.err,y)$coefficient[2])

plot(x.err, y.err, xlim=c(-3,3), ylim=c(-20,20))
abline(lsfit(x.err,y.err)$coefficient[1], lsfit(x.err,y.err)$coefficient[2])
```









Adjusting For Measurement Error - Frequentist Approach

As we have seen, there is not much one can do when there is measurement error in y. There is no bias, but the measurement error in y simply adds to the residual error, and the two effects become inseparable. The best thing to do is to measure y as accurately as possible, and simply live with the decreased precision, or increase the sample size. At least in this case there is no bias.

When there is measurement error in x the situation is more serious because effects will be biased towards zero, and no increase in sample size can help.

There is a theorem, however, which relates the β^* when $x.err$ is used to the true value we want to estimate, β:

Theorem 1: Let α and β be the true values of the linear regression between x and y, and let α^* and β^* be the biased values when x is measured with error and used in a regression with y, in other words, when $x.err$ is used. Further, let μ and σ^2 be the mean and variance of x in the population of interest, and let $\tau = \frac{sd(meas.x.err)}{sd(x)} = \frac{sd(meas.x.err)}{\sigma}$, where sd(meas.x.err) is the standard deviation of the measurement error around x. Then we have:

$$\begin{aligned} \alpha^* - \alpha &= \frac{\mu\beta}{1+\tau^2} \\ \frac{\beta^*}{\beta} &= \frac{1}{1+\tau^2} \end{aligned}$$

Unraveling these two equations to solve for our parameters of interest, we have:

$$\begin{aligned} \alpha &= \alpha^* - \frac{\mu\beta}{1+\tau^2} \\ \beta &= (1+\tau^2)\beta^* \end{aligned}$$

So, we can adjust for the measurement error through some relatively simple formulae.

Catch: In order to adjust for this measurement error, we need to know the true value of τ.

From a frequentist viewpoint, one can use a "best guess" value, and hope for the best. Any inferences, however, will not account for the extra uncertainty inherent in plugging in an "exact" value for τ when it is really uncertain.

We will soon see how the Bayesian solution uses a prior distribution for τ rather than an exact value, getting around this problem, so that final inferences include all uncertainty.

First, let's see what happens if we try this adjustment in our example:

From situation 3 (error in x but not in y), we had: $\alpha^* = 0.8346$, and $\beta^* = 0.4965$. Furthermore, we know (somewhat unrealistically, in real practice this is usually only known very approximately) that $\tau = \frac{sd(meas.x.err)}{sd(x)} = \frac{2}{1} = 2$ from the way the data were simulated. We will also need an estimate of the mean of x, but here we have an unbiased sample of x (because measurement error was unbiased), so can use

```
> mean(x.err)
[1] -0.1439378
```

as our estimate of μ. (The true value was of course 0.)

Plugging into the formulae, we get:

```
#  Adjusted value for slope beta

> (1 + 4)*0.4965
[1] 2.4825

#  Which is not too far from the true value of 2,
#  certainly better than the biased estimate of 0.4965

#  Adjusted value for intercept alpha
#  (using adjusted value for beta above)

> 0.8346 - (-0.1439378* 2.4825)/(1 + 4)
[1] 0.906

#  Which is quite close to the true value of 1,
#  improved over the biased estimate of 0.8346
```

So the method seems to work, provided one knows τ reasonably well.

What happens when there are other covariates, in addition to x? There is a second more complex theorem which describes this case, beyond the scope of this course.

Basic idea: Similar to the above theorem, but now we adjust β as follows:

$$\frac{\beta^*}{\beta} = \frac{1}{1 + \frac{\tau^2}{1-\rho^2}}$$

where ρ is the correlation coefficient between x and z, the additional covariate in the model, assumed measured without error.

Notice that if the new covariate z is uncorrelated with x, then $1 - \rho = 1$, and the formula reduces to the earlier one. However, there is high correlation, for example, if $\rho = 0.95$ or $\rho = -0.95$, for example, the formula (approximately) becomes

$$\frac{\beta^*}{\beta} = \frac{1}{1 + 10\tau^2}$$

so that

$$\beta = \beta^*(1 + 10\tau^2)$$

So, in the presence of correlated covariates, even if these new covariates are measured without error, the problem can be much worse.

In preparation for the next section, create and save some similar data as a text file in (close to) WinBUGS format.

We round all values to 3 decimal places for convenience.

```
> x <- round(rnorm(200),3)

> y <- round(1 + 2 * x + rnorm(200, mean=0, sd=4),3)

> x.err <- round(x + rnorm(200, mean=0, sd=2),3)

> error.dat <- data.frame(x, y, x.err)

> dput(error.dat, file="g:\\error.txt")

#  Some editing required after saving, delete row + column names,
#  and remove dim + structure parts.
```

```
#  Note:  can actually avoid this editing by using a "list"
#  rather than a data frame, as follows:

#  dat.list <- list(x=x, y=y, x.err=x.err)
#  dput(dat.list, file="g:\\error2.txt")

#  Much less editing then required.
```

Adjusting For Measurement Error - Bayesian Approach

If we know how to run a linear regression in WinBUGS, then it is very easy to add a measurement error component.

Here is a program that does both at the same time, and also calculates the degree of bias for both the slope and the intercept.

Note the data set has three repetitions of y, which is used three times, once for each model.

```
model
{

#   First run a model using no mismeasured variables

for (i in 1:200)
{
y.mean1[i] <- alpha1 + beta1*x[i] #  Regression model
 y1[i]   ~ dnorm(y.mean1[i],tau1) #  For x and y (no error)
 }
    alpha1    ~ dnorm(0 ,0.001)   #  Priors for variance + reg parms
    beta1     ~  dnorm(0 ,0.001)
    tau1      <- 1/(sigma1*sigma1)
    sigma1   ~ dunif(0,100)

#   Next run a model with x mismeasured, but no adjustment

 for (i in 1:200)
   {
y.mean2[i] <- alpha2 + beta2*x.err[i]  #  Regression model
y2[i]   ~ dnorm(y.mean2[i],tau2)       #  For x.err and y
```

```
    }
    alpha2    ~ dnorm(0 ,0.001)   #  Priors for variance + reg params
    beta2     ~  dnorm(0 ,0.001)
    tau2      <- 1/(sigma2*sigma2)
    sigma2   ~ dunif(0,100)

#   Finally, run a model with x mismeasured, and try to adjust

    for (i in 1:200)
   {
y.mean3[i] <- alpha3 + beta3*x.pop[i]
y3[i]   ~ dnorm(y.mean3[i],tau3)
x.err[i] ~ dnorm(x.pop[i], tau.error) # Error varies arouond true value of x
x.pop[i] ~ dnorm(0,1)                   #  Distribution of x in population
    }
alpha3    ~ dnorm(0 ,0.001) #  Priors for variance + reg parms
beta3     ~  dnorm(0 ,0.001)
tau3      <- 1/(sigma3*sigma3)
sigma3   ~ dunif(0,100)
tau.error <- 1/(sigma.error*sigma.error)  #  function of sigma, needed by WinBUGS
sigma.error ~ dunif(1.9, 2.1)        #  true value = 2, we add some uncertainty

#  Compare some estimates to check the bias in the intercepts and slopes

bias.slope <- beta2 - beta1
bias.slope.adj <- beta3 - beta1

bias.int <- alpha2 - alpha1
bias.int.adj <- alpha3 - alpha1

}

#  inits

list(alpha1 = 0, beta1 = 0,  sigma1 = 1,  alpha2 = 0,
   beta2 = 0,  sigma2 = 1, sigma.error=2, alpha3 = 0,
   beta3 = 0,  sigma3 = 1, sigma.error=2)


#  Data

list(x = c(-0.773, 0.573, -0.491, -0.976, 0.383, 0.843,
-0.563, 0.344, 0.936, 0.498, -0.535, -0.972, 0.245, 1.164, -0.284,
0.473, 0.884, -2.059, -0.301, 1.626, -0.278, 0.149, -0.891, 0.415,
-0.27, -0.392, -0.126, 1.597, -1.287, 0.422, 1.057, -1.145, 0.076,
```

```
-1.25, -0.198, -1.47, -2.362, -0.557, -0.778, -2.359, -0.974,
-0.537, -1.367, 0.364, -0.517, 1.325, 0.478, -0.203, 1.358, 0.302,
-0.216, 1.303, 0.775, 0.819, 0.131, -0.1, 0.089, 0.699, 0.231,
0.369, 1.766, -1.163, 1.102, 0.204, -1.745, 0.938, -0.1, -0.255,
0.169, -1.104, 0.454, -2.025, 0.542, -0.587, -1.572, -0.887,
0.312, 1.058, -0.632, -0.586, 0.858, -0.905, 1.423, 0.947, 1.545,
-0.57, -0.179, -0.279, -0.455, 0.629, 0.641, 1.447, -0.419, -1.012,
-0.653, 0.937, -0.027, -1.944, -1.335, -0.391, -0.911, -0.489,
1.663, -0.582, 0.359, -0.781, -1.688, 0.385, 0.195, 0.747, -1.05,
2.15, 1.302, 1.278, -0.514, -0.05, -0.706, 1.109, -0.693, -2.78,
-0.254, -0.574, -1.624, -0.709, 1.101, -0.688, 0.234, 2.092,
2.926, 0.59, 0.927, 0.307, 1.075, -0.394, -0.695, -0.403, -1.379,
-0.977, -0.944, 1.035, 0.943, 0.49, -0.891, -0.24, -0.099, -1.95,
-0.104, 0.416, 0.33, 0.168, -1.122, 0.624, -0.894, -0.029, -1.543,
0.396, 0.193, -0.767, 0.8, -1.108, -2.214, 1.798, 0.743, 0.044,
-1.487, 1.669, 1.694, 0.066, 0.766, -2.149, 1.925, -1.617, 1.363,
1.652, -0.645, -0.289, -0.59, 0.377, -0.846, -0.537, 1.021, 0.228,
-1.866, 0.435, -0.021, -1.979, -0.867, 1.988, 0.941, -0.765,
1.37, 0.488, 0.851, 1.05, -0.011, -0.453, 0.323, -1.146, 0.866,
2.982),
y1 = c(-3.465, -1.485, -1.905, 6.337, 0.622, 10.688, 0.515,
3, 1.372, 3.218, -2.892, -1.101, 10.143, 1.353, -2.498, 9.04,
2.769, -6.295, -1.027, 11.572, -2.445, -0.783, -3.605, -0.489,
3.19, -3.065, -2.773, 0.081, -8.955, -1.095, -1.325, -0.062,
-5.591, -4.711, 4.931, 1.699, -1.202, 6.872, -4.775, -5.069,
-2.717, -0.732, 0.877, 2.233, 5.842, -0.131, 10.811, -5.492,
1.628, 7.701, 1.761, 10.032, 3.246, -1.092, 5.543, 3.134, 8.765,
1.199, -2.659, 3.1, 0.82, -10.346, 2.738, 8.096, -6.486, 0.318,
4.004, 6.167, -2.695, -5.096, 6.562, 0.746, 0.214, 8.739, -5.835,
-1.846, 4.513, 1.773, 0.985, -6.623, -1.854, 0.262, -0.471, 12.464,
13.132, -0.346, 4.674, -0.447, 3.296, 0.61, -5.121, 7.239, -7.117,
-4.076, -0.219, 6.998, 0.01, -5.798, -8.511, -4.167, -0.845,
-0.134, 11.05, 3.284, 2.606, 0.611, 1.822, 1.647, 0.507, 5.479,
10.602, 7.748, 10.343, -2.352, 2.88, 3.532, 7.353, -0.523, 1.9,
-2.061, 1.917, 0.086, -0.123, 1.232, 1.811, 2.234, 0.043, 6.909,
11.925, 1.05, 3.695, 2.264, 4.318, -0.525, -5.137, 0.084, 0.247,
-1.218, 1.619, -2.54, 7.639, 5.781, 2.449, -0.613, 1.543, 0.355,
1.178, 6.464, -1.166, 2.444, -2.335, 0.273, -6.055, 2.278, -4.06,
-2.199, -2.711, -2.094, -6.178, -0.503, -8.925, -3.119, 14.824,
-0.685, -6.189, 10.608, -0.217, 0.097, 1.725, -0.866, -0.245,
-7.874, 11.753, 6.9, -0.026, -1.161, -5.549, -0.742, -3.55, 1.285,
-1.462, 2.631, -9.482, 2.319, -9.871, -0.559, 1.219, 12.237,
-5.319, -7.45, 6.72, 5.41, 9.021, 3.332, -2.789, -2.699, 2.317,
2.822, 2.571, 3.646),
y2 = c(-3.465, -1.485, -1.905, 6.337, 0.622, 10.688, 0.515,
```

```
3, 1.372, 3.218, -2.892, -1.101, 10.143, 1.353, -2.498, 9.04,
2.769, -6.295, -1.027, 11.572, -2.445, -0.783, -3.605, -0.489,
3.19, -3.065, -2.773, 0.081, -8.955, -1.095, -1.325, -0.062,
-5.591, -4.711, 4.931, 1.699, -1.202, 6.872, -4.775, -5.069,
-2.717, -0.732, 0.877, 2.233, 5.842, -0.131, 10.811, -5.492,
1.628, 7.701, 1.761, 10.032, 3.246, -1.092, 5.543, 3.134, 8.765,
1.199, -2.659, 3.1, 0.82, -10.346, 2.738, 8.096, -6.486, 0.318,
4.004, 6.167, -2.695, -5.096, 6.562, 0.746, 0.214, 8.739, -5.835,
-1.846, 4.513, 1.773, 0.985, -6.623, -1.854, 0.262, -0.471, 12.464,
13.132, -0.346, 4.674, -0.447, 3.296, 0.61, -5.121, 7.239, -7.117,
-4.076, -0.219, 6.998, 0.01, -5.798, -8.511, -4.167, -0.845,
-0.134, 11.05, 3.284, 2.606, 0.611, 1.822, 1.647, 0.507, 5.479,
10.602, 7.748, 10.343, -2.352, 2.88, 3.532, 7.353, -0.523, 1.9,
-2.061, 1.917, 0.086, -0.123, 1.232, 1.811, 2.234, 0.043, 6.909,
11.925, 1.05, 3.695, 2.264, 4.318, -0.525, -5.137, 0.084, 0.247,
-1.218, 1.619, -2.54, 7.639, 5.781, 2.449, -0.613, 1.543, 0.355,
1.178, 6.464, -1.166, 2.444, -2.335, 0.273, -6.055, 2.278, -4.06,
-2.199, -2.711, -2.094, -6.178, -0.503, -8.925, -3.119, 14.824,
-0.685, -6.189, 10.608, -0.217, 0.097, 1.725, -0.866, -0.245,
-7.874, 11.753, 6.9, -0.026, -1.161, -5.549, -0.742, -3.55, 1.285,
-1.462, 2.631, -9.482, 2.319, -9.871, -0.559, 1.219, 12.237,
-5.319, -7.45, 6.72, 5.41, 9.021, 3.332, -2.789, -2.699, 2.317,
2.822, 2.571, 3.646),
y3 = c(-3.465, -1.485, -1.905, 6.337, 0.622, 10.688, 0.515,
3, 1.372, 3.218, -2.892, -1.101, 10.143, 1.353, -2.498, 9.04,
2.769, -6.295, -1.027, 11.572, -2.445, -0.783, -3.605, -0.489,
3.19, -3.065, -2.773, 0.081, -8.955, -1.095, -1.325, -0.062,
-5.591, -4.711, 4.931, 1.699, -1.202, 6.872, -4.775, -5.069,
-2.717, -0.732, 0.877, 2.233, 5.842, -0.131, 10.811, -5.492,
1.628, 7.701, 1.761, 10.032, 3.246, -1.092, 5.543, 3.134, 8.765,
1.199, -2.659, 3.1, 0.82, -10.346, 2.738, 8.096, -6.486, 0.318,
4.004, 6.167, -2.695, -5.096, 6.562, 0.746, 0.214, 8.739, -5.835,
-1.846, 4.513, 1.773, 0.985, -6.623, -1.854, 0.262, -0.471, 12.464,
13.132, -0.346, 4.674, -0.447, 3.296, 0.61, -5.121, 7.239, -7.117,
-4.076, -0.219, 6.998, 0.01, -5.798, -8.511, -4.167, -0.845,
-0.134, 11.05, 3.284, 2.606, 0.611, 1.822, 1.647, 0.507, 5.479,
10.602, 7.748, 10.343, -2.352, 2.88, 3.532, 7.353, -0.523, 1.9,
-2.061, 1.917, 0.086, -0.123, 1.232, 1.811, 2.234, 0.043, 6.909,
11.925, 1.05, 3.695, 2.264, 4.318, -0.525, -5.137, 0.084, 0.247,
-1.218, 1.619, -2.54, 7.639, 5.781, 2.449, -0.613, 1.543, 0.355,
1.178, 6.464, -1.166, 2.444, -2.335, 0.273, -6.055, 2.278, -4.06,
-2.199, -2.711, -2.094, -6.178, -0.503, -8.925, -3.119, 14.824,
-0.685, -6.189, 10.608, -0.217, 0.097, 1.725, -0.866, -0.245,
-7.874, 11.753, 6.9, -0.026, -1.161, -5.549, -0.742, -3.55, 1.285,
-1.462, 2.631, -9.482, 2.319, -9.871, -0.559, 1.219, 12.237,
```

```
-5.319, -7.45, 6.72, 5.41, 9.021, 3.332, -2.789, -2.699, 2.317,
2.822, 2.571, 3.646),
x.err = c(-3.068, 0.494, -1.219, 2.298,
-1.813, 2.241, -0.927, 1.3, 3.924, 0.926, -2.259, 4.019, 0.855,
-0.352, -0.24, 0.578, 0.162, 0.342, -0.283, 2.727, -1.334, -1.882,
1.071, 0.713, 0.101, 2.21, 1.23, 3.613, -2.157, 1.014, 1.552,
-1.685, -1.225, -1.272, -0.102, -0.112, 0.79, 2.162, -2.278,
-2.478, -1.469, -2.665, -1.488, -0.712, 0.38, 0.143, 3.288, 0.789,
-1.696, 0.958, -0.787, 0.479, -0.438, 2.507, 1.199, -3.121, 0.437,
0.684, 0.629, -0.76, 2.355, -3.818, 3.037, 0.609, -2.155, -0.576,
-1.304, -0.16, -1.455, 0.352, -1.335, -2.524, 2.111, -1.286,
0.353, 0.81, 2.7, 4.207, 1.494, 2.967, 0.206, 1.416, 0.063, -0.068,
-1.209, 2.114, -1.263, -1.81, 0.334, 2.535, 3.299, 0.293, -0.689,
0.074, -1.601, 3.213, 2.329, 0.422, 0.624, -1.95, 0.438, -0.308,
1.452, -0.297, -0.089, -4.316, -1.02, -1.111, 0.611, 1.812, 1.694,
2.373, 0.495, 4.449, -1.212, -2.328, -4.378, 3.292, 0.306, -2.975,
-3.059, -3.275, -1.835, -2.992, 0.198, 2.097, 0.61, 2.276, 1.347,
-0.4, -0.506, -4.684, 1.626, -0.924, -1.06, -3.898, -1.075, -3.005,
-2.083, -0.982, -0.963, -2.236, 1.255, 1.856, 0.637, 1.494, 0.854,
1.278, -0.344, 0.44, 0.4, -1.581, 0.071, 0.215, -3.986, -1.071,
1.506, 0.066, 4.873, 0.686, -2.128, -0.833, 4.565, 3.217, -2.734,
0.307, 3.649, 0.785, 3.153, -2.777, 2.39, -1.659, -0.584, 0.264,
-0.269, 0.73, 2.698, -3.404, 0.179, -3.658, -0.479, -3.179, 1.717,
3.114, 2.326, -2.47, -1.021, 1.345, 1.478, -2.357, 2.324, -0.55,
-0.964, -1.255, -0.456, 1.438, 3.682, -1.464, -1.546, 2.119))

#  Results
```

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
alpha1	1.066	0.3086	0.003118	0.4684	1.066	1.682	1001	10000
alpha2	0.9804	0.3464	0.003088	0.3057	0.9873	1.649	1001	10000
alpha3	0.9767	0.345	0.003895	0.2925	0.9781	1.648	1001	10000
beta1	2.376	0.2921	0.003182	1.804	2.376	2.95	1001	10000
beta2	0.4296	0.1753	0.001811	0.089	0.4279	0.7716	1001	10000
beta3	2.134	0.8251	0.03023	0.4845	2.156	3.715	1001	10000
bias.int	-0.0852	0.4647	0.004184	-0.9892	-0.07837	0.81	1001	10000
bias.int.adj	-0.08885	0.4642	0.004917	-1.006	-0.08975	0.8234	1001	10000
bias.slope	-1.946	0.3385	0.003757	-2.612	-1.947	-1.283	1001	10000
bias.slope.adj	-0.2413	0.8751	0.03027	-1.975	-0.2193	1.423	1001	10000
sigma1	4.332	0.2202	0.002178	3.933	4.323	4.792	1001	10000
sigma2	4.939	0.2494	0.002472	4.482	4.927	5.447	1001	10000
sigma3	4.489	0.4135	0.01425	3.583	4.518	5.228	1001	10000

Notice the almost perfect adjustment after measurement error correction.

Final Conclusions

- Measurement error is an important and often brushed aside topic. It arises in practically every epidemiological study.
- We have just scratched the surface, many books exist on measurement error.
- Given large bias that can arise, try to minimize error at the design stage.
- If that is not possible, try to adjust via measurement error adjusting models.
- Bayesian methods more flexible than frequentist methods, and easy to program even complex equations using WinBUGS.
- The case with logistic regression is theoretically more complicated. Bias can be in either direct, away or towards the null value.
- Bayesian logistic regression measurement error adjustment models can be run in exactly the same way as linear regression, using WinBUGS. However, these methods are less straightforward from a frequentist viewpoint.