Code ▾

# Week 8 Notes Part 2 Extras

Author: Brendan Gongol

Last update: 09 January, 2023

## 1 Overview

In this Note we look into a somewhat eclectic collection of a few additional methods and approaches. By no means are these methods uninteresting. On the contrary, they are important and widely used, but we simply do not have enough time in this course to cover them in all the detail in addition to the fundamental, basic approaches we have been learning about. Due to the importance of these methods, we will still describe them briefly if only to give you an introduction and initial idea of what they are used for and what they are about. If you ever need to use these methods and/or to understand them better, this will hopefully give you a head start. Despite giving only a cursory introduction into these methods, the Note will also show some R code that you can use in practice in order to perform a simple analysis (or solve a simple homework problem!).

## 2 Random Effects Models

In statistics and experimental design contexts, effects are the factors the outcome variable depends upon. The fixed effects are those that are part of the study design and are controlled (measured) in the experiment (so they are not by any mean fixed values). Studying and contrasting contributions from the fixed effects towards the outcome is among the main goals of an experiment. The examples of fixed effects are male vs female, control vs drug treatment, subject's genetic background, etc. In addition to fixed effects, there are often (always?) factors (effects) that contribute to the variance of the outcome (i.e. cause some changes in the outcome), but cannot be directly controlled and/or are not part of the study design. Namely, there can be differences between the sample batches (reagents or individual machine, e.g. sequencer or microarray scanner, used to process sample batches may differ; they could be handled by different technicians etc.); there can be (uncontrollable) differences between individual patients, and so on. These are referred to as random effects.

Note that while convenient, this definition is not absolute and is somewhat "philosophical". The distinction between fixed and random effects is made, at least to a great extent, depending on the study design and goals. For instance, genetic background is a fixed effect if it is specifically assessed in a study and we are asking if an $A \rightarrow C$ mutation in some gene is associated with a greater predisposition to a particular disease; however if we choose patients for the study based on other factors and do not check for their individual genotypes, then genetic variation (which may certainly affect many outcomes, such as blood pressure, cholesterol level, gene expression levels, even response to drugs) becomes an uncontrollable patient-to-patient variability (what we will see as "random variation" or "noise" in our experiment), i.e. a random effect. Similarly, while batch effect is an unwanted random effect in most studies, it can be a primary goal of a technology development and quality assurance effort in a production lab, in which case reagents and their shelf time, as well as individual pieces of equipment used are carefully controlled, so they become fixed effects.

One way of looking at and distinguishing fixed vs random effects is to consider what would happen in a larger study with the same design: levels of fixed effects will not change (male/female, control/drug, etc), while levels of random effects will change (more sample batches, more patient genetic backgrounds). This is of course just another way of specifyingwhat we control (i.e. measure and intelligently let into/leave out of the study) vs what we do not.

Since the distinction is quite philosophical as we can see, and we are not interested that much in the philosophy but rather in the data and what we can do with them, we will take fixed and random effects for what they are: convenient labels we can use to refer to different factors with regard to a specific experimental design. The true value of the distinction is realized, of course, only when we can build a better model that helps us get a more accurate view of the data. To that end, let us consider a few of examples:

- We are troubleshooting a chemical synthesis process. The synthesis is performed in multiple batches, $i = 1...M$, and in each batch we perform several repeated measurements, $j = 1...N$, of molecular weight of the compound(s) or generate replicated mass-spectrograms, and so on.
- We measure gene expression, and in our study we use multiple animals, $i = 1...M$, and generate multiple technical replicates (e.g. run several microarrays), $j = 1...N$, for each animal
- We are collecting data for the study from multiple participating hospitals, $i = 1...M$, and each hospital reports cholesterol levels (and hopefully many other parameters) for patients $j = 1...N$ enrolled in the study at that site

Note that in all the above cases when we list effects and make a distinction between fixed and random ones, we do make some assumptions about the effect size hierarchy. For instance, setting up a multiple hospital problem the way we did makes sense if we have a reason to believe that "systematic" hospital-to-hospital variance is large so it is an important effect that needs to be specifically taken into account and modeled.

In the cases like the ones described above, one can try fitting the following model:

**(1)**

$$y_{ij} = \mu + \alpha_i + \epsilon_{ij}$$

where $\mu$ is the grand mean (can be within a specific group defined by levels of fixed effects, e.g. mean cholesterol level of healthy male patients within specific age group), $i$ enumerates batches, subjects, hospitals (or anything we are trying to model as a random effect), j enumerates replicated measurements within a batch or patients coming from the same hospital source site, etc – depending on the problem at hand and particular breakdown into fixed/random effects; we separately model random effect as $\alpha_i - N(0, \sigma_r)$ (contribution specific to i-th batch, animal, hospital in the above examples, and we usually assume it is normally distributed), and the remaining variance (that we assume to be completely random, structureless and homogeneous) is represented by a random variable $\epsilon_{ij} - N(0, \sigma)$.

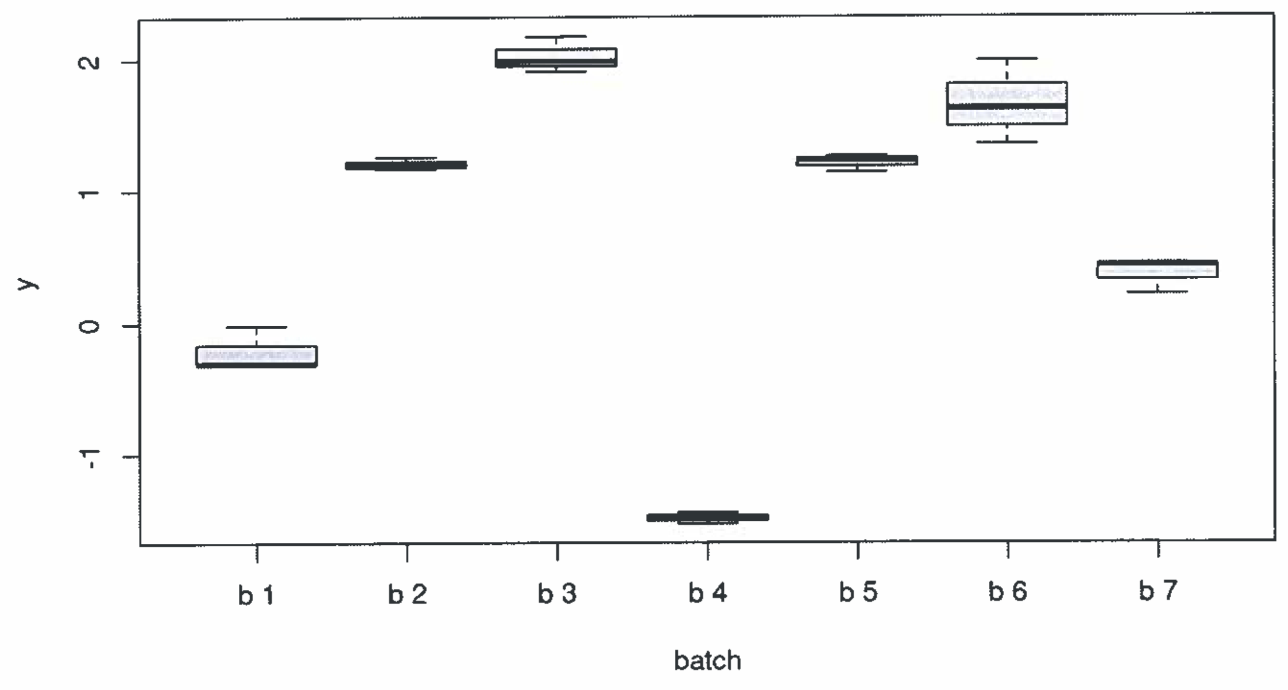
The purpose of using models with more complex structure as shown above is to explicitly account for random effects ("batch effects") in the hope of getting more accurate estimate of sample statistic (such as $\mu$). There is indeed some information in the data that can be used in order to fit a model of the type shown in

Eq.(1). Note that since the same contribution (systematic "offset") $\alpha_i$ goes into multiple $y_{ij}$ values for different $j$, the observed values of $y$ will exhibit partial correlation, which can be used to infer distribution of $\alpha$ (the random observations of noise component $\varepsilon$ are by definition uncorrelated).

We do not have room in this course to further discuss how the random effects models are built and evaluated, but we will consider instead a brief example in R (you will need to download and install `lme4` library):

Hide

```
set.seed(1234)
n.batch <- 7 # number of batches
n.reps <- 3 # n replicated measurements in each batch
sd.batch <- 1.0 # sd of random effect term (sigma_r)
sd.rep <- 0.2 # sd of measurement error, epsilon
mu <- 1.0 # grand mean
# now we will generate 21 measurements: 3 per batch, 7 batches:
# sample 7 distinct values for alpha_i; replicate 3 times:
batch.eff <- rep(rnorm(n.batch,sd=sd.batch),n.reps) # alpha-term
rep.eff <- rnorm(n.reps*n.batch,sd=sd.rep) # sample measurement err
re.df <- data.frame(batch=rep(paste("b",1:n.batch),n.reps),
 y=mu+batch.eff+rep.eff) # make sure everything in matching order
boxplot(y~batch,re.df)
library(lme4)
```



Hide

```
lmer(y~(1|batch),re.df)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: y ~ (1 | batch)
##    Data: re.df
## REML criterion at convergence: 17.1974
## Random effects:
##  Groups   Name        Std.Dev.
##  batch    (Intercept) 1.2131
##  Residual             0.1587
## Number of obs: 21, groups:  batch, 7
## Fixed Effects:
## (Intercept)
##      0.6803
```

In the code shown above, we simulate the data as follows: we create 7 batches of observations, 3 observations in each. The standard deviation of our measurements is assumed to be 0.2 (epsilon term in the Eq.(1)), and the standard deviation of the batch-to-batch variation (random effect, the alpha term) is set to 1. We then generate 21=7x3 measurements by sampling from the corresponding normal distributions and save them into a dataframe alongside with the batch label. The next `boxplot()` command generates the figure shown below: this is the distribution of our simulated data stratified by batch. Next we load the random-effects library `lme4` and fit the model. Note the formula term: we use the simplest model here where we specify that the outcome variable $y$ does not depend on any (fixed effect) explanatory variable, since we did not build any such variable into our simulation (within each batch, $y$ is just normally distributed without further association with anything else). Hence we simply use the term "1" in the formula and thus look only for the mean of $y$, similar to the "conventional" linear model with formula $y \sim 1$. However, our observations are now "conditioned" on the batch (random effect). The summary of the fitted model shows that despite the huge batch-to-batch variation (see the plot below), the model still did a decent job in several respects:

- The fitted standard deviation attributed to batch (random effect) is ~1.2, close to the value of 1 we built into our simulation;
- The standard deviation of the "residual" (i.e unexplained variance=measurement error) is ~0.15, close to the 0.2 measurement error we built into the model.
- The grand mean of the whole sample is 0.68, which is what you would get if you computed `mean(re.df$y)` directly – we cannot do better than the available data tell us, even if we know that there is strong random effect involved. However, the standard error of that mean is estimated as ~0.46 (so that the actual value `μ=1` we built into the simulated data is well within one standard error from the estimate); if we were not aware of the random effects and were not using random effects model to fit the data, we would estimate the standard error of the mean as `sd(re.df$y)/sqrt(21)`, which would give us only ~0.25. In other words, the random effects model was able to detect the batch effect and adjust the standard error of the mean accordingly (due to large systematic batch-to-batch variation it is greater than it would be if we had just 21 structureless, normally distributed observations with large variance). In other words, we cannot improve the mean beyond what the data tell us, but we are able to tell that the data are "bad" and the mean is very imprecise.

## 3 Generalized Linear Models

The linear models we have been studying so far are in fact quite flexible: as we discussed, they are "linear" only with respect to the model coefficients, while the explanatory random variables can undergo arbitrary transformations. In generalized linear models, the limitation on the left hand side of the model equation is further lifted: it is possible to look for the dependence of an arbitrarily transformed response variable $Y$, $f(Y)$, on the explanatory variables. In other words, with generalized linear models we can fit dependencies of the form:

**(2)**

$$f(Y) = a_o + a_1\varphi_1(X_1) + \ldots + a_n\varphi_n(X_n) + \epsilon$$

where the right hand side is a usual linear model (arbitrarily transformed explanatory random variables, linear w.r.t model coefficients), and left hand side generalizes linear model by introducing an arbitrary link function, f. Additionally, generalized linear models can allow for non-normally distributed errors (residuals).

Here we will consider just one very important example of generalized linear models: logistic regression. The goal of this model is to fit a categorical outcome variable against (continuous) explanatory variables. Furthermore, we will be considering only a binary categorical outcome (e.g. disease vs healthy, success vs failure, disease relapse vs disease-free), although generalizations do exist. Let us assume, for the sake of certainty, that we are trying to predict (the chances of) disease relapse based on gene expression level of specific gene(s). The model is set up as follows:

- We are still in the statistical, probabilistic domain: no measured gene expression level can probably predict the outcome with certainty. Even though we observe discreet events (relapse/no relapse), they are realizations of a Bernoulli process ("coin toss") with unknown and unobserved probability $P$. We do expect this probability to depend on the observed expression level(s): at some expression levels relapse is more likely than at others (that's why we fit the model in the first place – in order to see if such dependence does exist and to quantify it with the fitted model). Thus we can only hope to predict the probability, $P$, of the outcome (since the outcome is binary, the other level has probability $1 - P$ – it's a Bernoulli process). Of course, in some cases, when the evidence is overwhelming, we can have $P = 1$ or $P = 0$, at least in principle, but that's not a general case. Hence, our model should essentially state that the probability of the outcome is a function of explanatory variables, $P(X_1, \ldots, X_n)$: in certain ranges of gene expression levels the probability of relapse is higher than in the others.

- You can recognize by now, that if we had the probability P measured directly and extensively at different combinations of values of the explanatory variables, we could try simply fitting a familiar linear model $P \sim X_1 + X_2 + \ldots + X_n$ (or we could include interactions if we felt like it – this is not the point). There are two problems, however:
    - The empirical probability function $P$ that we want to fit (i.e. the "measured values") is not known. What we have are measured binary outcomes (disease relapse either happened or not in any particular patient). Strictly speaking, the probability function is not measurable at all in the case of continuous explanatory variables: we can never achieve finite density of patients at any expression level value coming from a continuous spectrum, so we cannot estimate the point probability density of a relapse precisely. Of course, we can approximate $P$ by looking at coarse-grained representation such as a histogram, similar to how we always represented empirical distributions (we will try this later as a learning example); we can even smoothen it out using somespline/kernel method. Alternatively, we can try fitting a conventional linear model directly to an outcome variable that happens to take only values 0 and 1. The machinery will formally work in such settings, but the results will be suboptimal. We hope we can do better if we assume some reasonable model for $P$ a priori.
    - The naïve linear model fit as shown above is unbounded: in the simplest possible case, $P = ax + b$, the predicted "probability" $P$ can clearly take n=both arbitrarily large values and negative values, depending on the measured expression value, $x$. Even if we use some

transformations of the explanatory variables (as in the right-hand-side of Eq.(2)), it is very difficult if possible at all to choose them properly and consistently in such a way that the predicted probability always satisfies the natural constraints $0 \leq P \leq 1$, and even if we achieve that, the model will be very cumbersome. Note that this problem occurs regardless of what data we use for P: we can fit a model to good experimental data (0/1 outcomes, or even estimate of the "true" function $P$ that satisfies all the constraints - assuming we could get it), but the predicted values from the fitted model will likely run out of $[0, 1]$ interval when expression values vary in a sufficiently broad interval.

- Logistic regression provides one possible solution to these problems by assuming the following functional dependence for $P$ (known as logistic function, which is found in many applications):

**(3)**

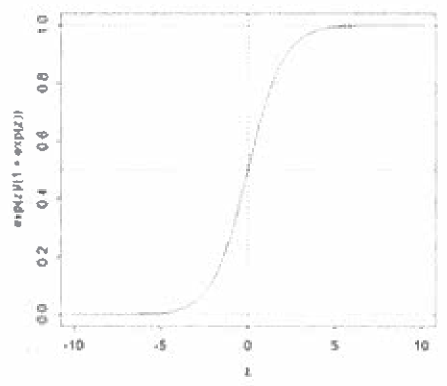
$$P(Z) = \frac{e^z}{1 + e^z}$$

It is the variable $Z$ that is governed by a linear model on the explanatory variables:

**(4)**

$$Z = a_o + a_1\varphi_1(X_1) + \ldots + a_n\varphi_n(X_n) + \epsilon$$

The plot of the function (3) is shown below:



We can see that while $Z$ can be unbounded, the probability $P(Z)$ is nicely bounded between 0 and 1 and transitions smoothly between the two limits when $Z$ changes from negative infinity to infinity (and it is this latter change that we fit as a linear model on the explanatory variables $X$). We can rewrite equations (3) and (4) by expressing $Z$ via $P$,

**(5)**

$$Z = log(\frac{P}{1 - P})$$

and substituting the expression (5) for $Z$ into (4). The model we are fitting then takes the following final form:

**(6)**

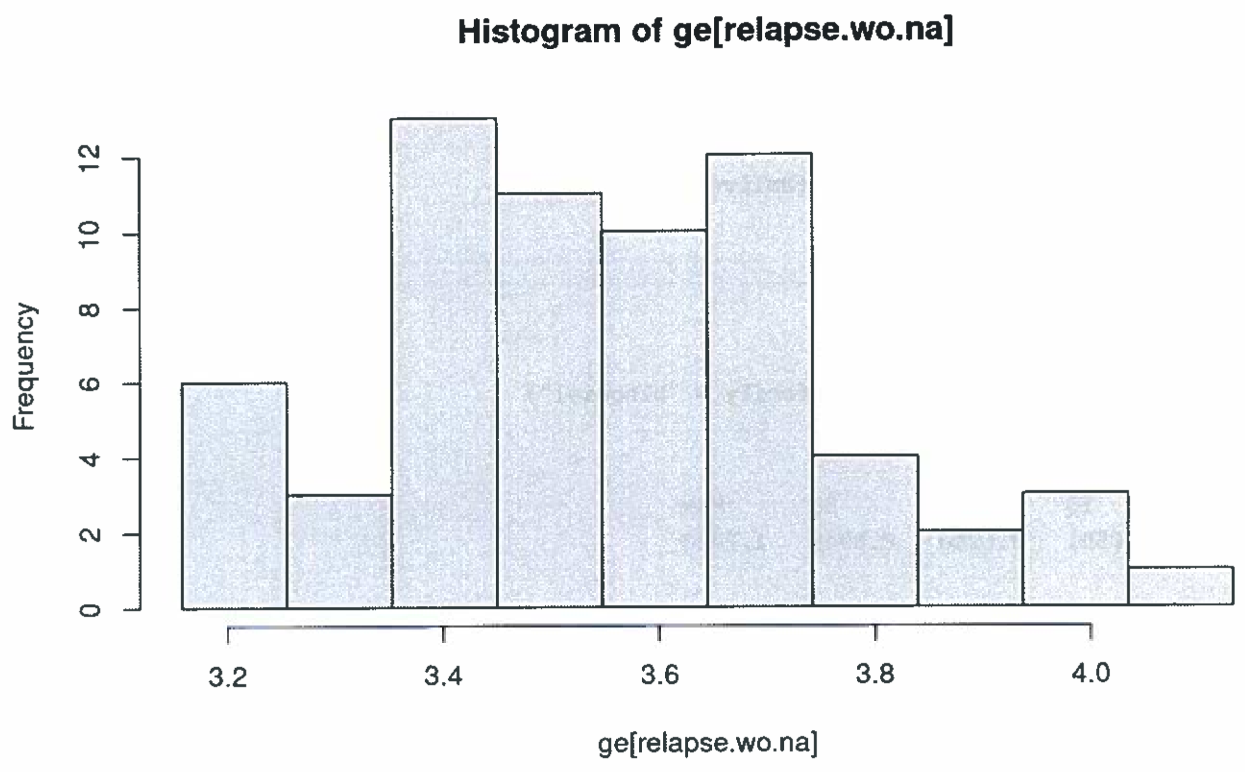
$$log(\frac{P}{(1 - P)}) = a_o + a_1\varphi_1(\chi_1) + \ldots + a_n\varphi_n(\chi_n) + \epsilon$$

Note that the equation (6) is exactly the equation of a generalized linear model (2) with specific link function $f(Y) = log(Y/(1-Y))$ (known as logit function).

It turns out that we can work out a formalism such that we can further feed such model with discreet outcomes 0/1 of the dependent variable, and the approximation for (directly unmeasurable) $P$ will be achieved automatically (in a maximum likelihood sense), so we can fit the generalized model directly on the categorical outcomes. We do not have time or space left to see how the derivation works, but let us see how we can work with real data in R:

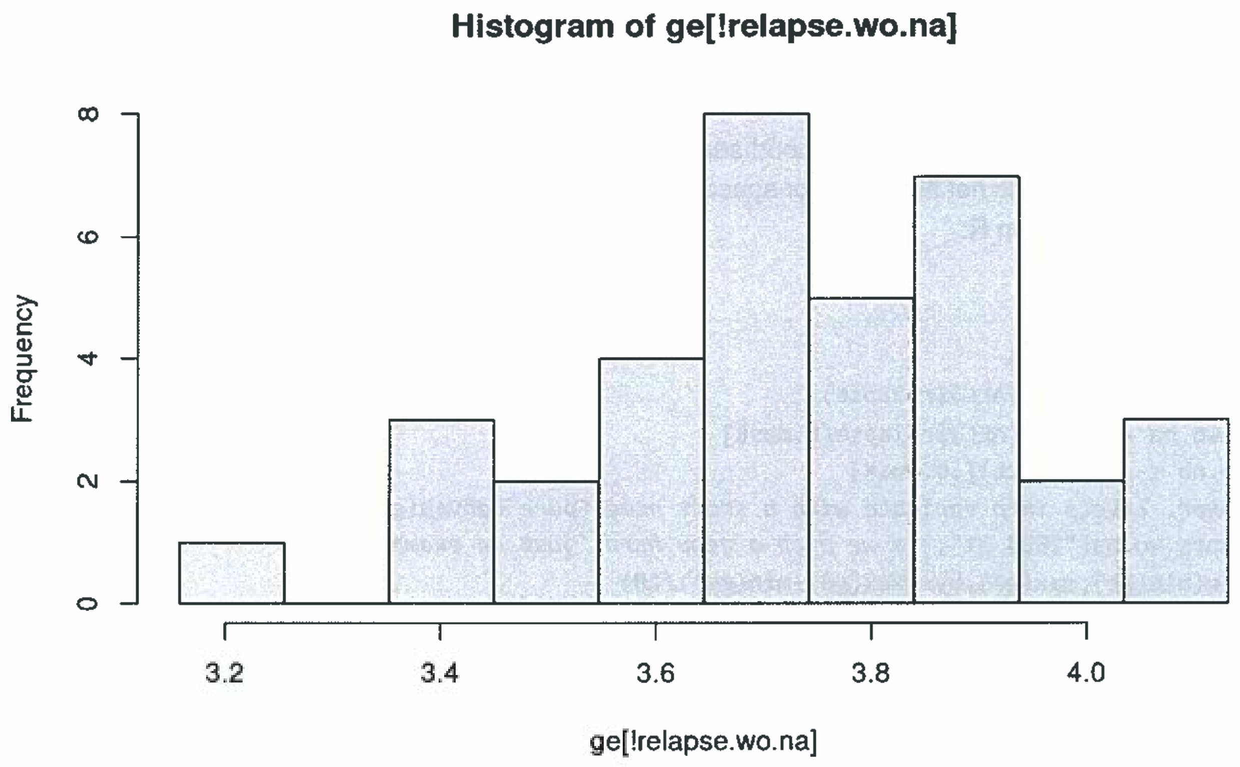
Hide

```
library(ALL); data(ALL)
b.mask <- !is.na(pData(ALL)$relapse)
relapse.wo.na <- pData(ALL)$relapse[b.mask]
exprs.wo.na <- exprs(ALL)[,b.mask]
# save expr. levels into variable with a short name (pure convenience):
ge <- exprs.wo.na["1584_at",] # we pick a gene here, just an example
br <- seq(min(ge),max(ge),by=(max(ge)-min(ge))/10)
cnt.1 <- hist(ge[relapse.wo.na],breaks=br)$counts
```



Hide

```
cnt.0 <- hist(ge[!relapse.wo.na],breaks=br)$counts
```



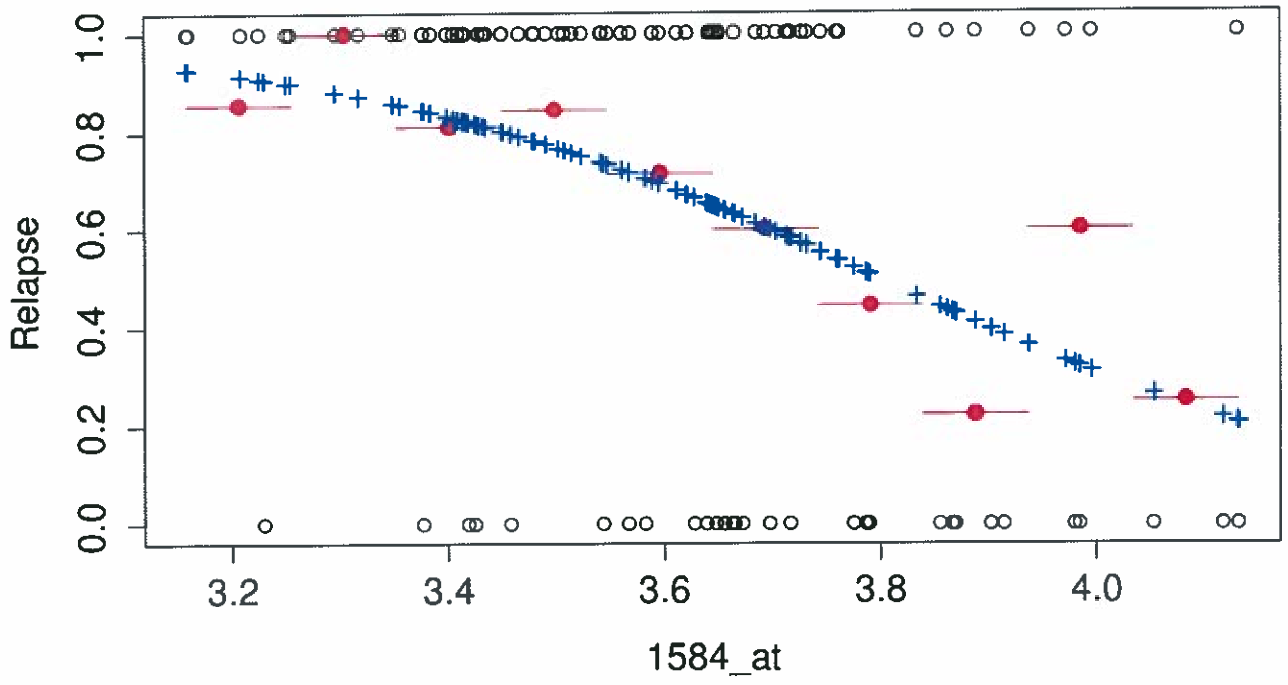
Hide

```
glm.1584.at <- glm(relapse.wo.na~ge, family="binomial")
summary(glm.1584.at)
```

```
##
## Call:
## glm(formula = relapse.wo.na ~ ge, family = "binomial")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1786  -1.0591   0.6210   0.8945   1.7720
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   15.228      4.199   3.626 0.000288 ***
## ge            -4.010      1.145  -3.502 0.000462 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 129.49  on 99  degrees of freedom
## Residual deviance: 114.57  on 98  degrees of freedom
## AIC: 118.57
##
## Number of Fisher Scoring iterations: 3
```

Hide

```
old.par<-par(ps=16)
plot(ge,relapse.wo.na, xlab="1584_at",ylab="Relapse")
points((br[1:10]+br[2:11])/2,
 cnt.1/(cnt.0+cnt.1),pch=19,col='red',cex=1.2)
segments(br[1:10], cnt.1/(cnt.0+cnt.1),
br[2:11], cnt.1/(cnt.0+cnt.1),col='red')
y.pred <- predict(glm.1584.at)
points(ge, exp(y.pred)/(1+exp(y.pred)),col="blue",pch='+')
```



Hide

```
par(old.par)
```

In the code fragment above, we initialize variables for relapse status and gene expression levels (omitting patients with no data available), and then immediately fit the generalized linear model for a specific gene (1584_at) using `glm()` function. Note the function interface:

- We specify the formula in the same way as we would for a "conventional" linear model: i.e. we are saying that we are fitting relapse ~ expression.
- However we are adding family argument: `glm()` is very flexible and without going into too much detail we will just mention that some commonly used error distributions and link functions are pre-defined in the context of this function and can be invoked by simply specifying their symbolic name. In this particular case, we instruct `glm()` to use default "binomial" family, which amounts to using (a) binomial error distribution (because we have a binary outcome variable) and (b) logit link function as in Eq. (5). according to Eq. (6). All the required transformations are performed internally.

- Note the convenience of the interface: the `glm()` call in our example will take directly measured binary outcome variable values from `relapse.wo.na` (as instructed by the formula), but perform the fit against gene expression levels according to (6), using the logit link function, all the required transformations are performed internally.

the plot of black circles only. ↑

the red + blue is done below.

no plot included in notes

The summary of our fit shows that it is quite significant, i.e. the likelihood of the relapse does seem to depend on the expression level of the gene we chose. $p: 0.00462$

In order to visualize the fit, we first plot the values of relapse variable ( `TRUE/FALSE` , interpreted as 1/0) against gene expression level (black dots in the figure below). This is not very informative, since we are looking at the realized binary outcomes, not the underlying probability of a relapse at any given gene expression level. One can glean some information from the density of the black dots: it does appear that at low expression levels the `relapse=TRUE` dots (at y=1 level) are placed much denser than `relapse=FALSE` ones (y=0). At high expression levels, the situation is opposite: there are somewhat more observations with `relapse=FALSE` than with `relapse=TRUE` . Thus we can conclude that the underlying chance of relapse is higher at low expression levels of gene (Affy probe, actually) 1584_at, and lower at high expression levels.

To confirm this observation, we take a more detailed look at the amounts of relapsed cases as function of gene expression. Since we have finite number of observations and expression level is a continuous variable, the only quick and feasible procedure is to artificially introduce some granularity into the data. As the code above shows, we break the range of observed expression levels into 10 equal intervals, and then use the function `hist()` in order to count how many patients (expression level observations) with `relapse=TRUE` and how many patients with `relapse=FALSE` fall into each such discretized interval. The counts per interval are assigned to the vectors `cnt.1` and `cnt.0` , respectively. Now that we have finite counts of relapse, r, and no-relapse, n, cases in each interval, we can estimate the probability of relapse for patients exhibiting expression levels in that interval as $P = r/(n+r)$. This is precisely the "granular" estimate of the underlying probability $P$ that we discussed above, in connection with (very suboptimal) possibility of fitting it directly with the model $P \sim ax + b$. Here we do not perform such a fit, of course, but just plot the estimated $P$ with the next `points()` command in the code fragment (thick solid red dots in the plot below). Each point is drawn at the middle of the corresponding expression level interval, and the span of the interval is further shown with horizontal red line (the next `segment()` command in the code).

The estimate we just did confirms our initial theory: there seems to be a clear trend for decreasing relapse probability with increasing expression levels of gene 1584_at. We should not read much more into this simple estimate since it suffers from small counts of data points (either relapse or no-relapse, or both) in some intervals, hence we should expect high levels of random fluctuations (errors) due to sampling. Also, the naïve procedure we are using suffers from boundary effects (especially in the intervals with low counts): with fixed-boundary intervals without any smoothing, a single data point located very close to the boundary can introduce very large difference between adjacent estimates, while intuitively it should give partial contributions to both intervals: look for example at the first, second, and third leftmost intervals in the plot below.

Finally, we draw the fit generated by our logistic regression model. In order to do that, we generate predicted values from the fitted model object (using the same `predict()` function as we used for "conventional" linear model). NOTE that:

- Generalized linear model implementation in R returns the values of $Z$ (Eq.(4)), not P, as "predicted" values. This makes sense if you remember that you can use different link functions with `glm()` , so what's returned is the most generic answer. You can then apply your specific link function manually to transform the data the way you need. Thus, in order to generate predicted probability values we manually transform $Z$ into Pusing logistic function, Eq.(3), in the last plotting command ( `points()` ).

As we can see from the final plot, at least qualitatively, the logistic regression provides a very decent fit (blue crosses) consistent with our rough interval-based estimate (red dots) of the relapse probability. The discrepancy between the estimates and the fitted line ("residuals" of our generalized linear model fit – although precise calculation of the residuals does not involve computation of "granularized" probability estimates, of course) is likely due to sampling fluctuations – note that we have larger discrepancy in the intervals where we have but a few data points.

## 4 Survival Analysis

The last important topic we introduce without proof this week is Survival Analysis. In this type of analysis the goal is modeling the waiting time to event (time-to-remission or time-to-death would be textbook examples, the latter one also being the reason for the "survival" in the method's name). In general discussion hereafter, we will be referring to this waiting time as "survival time".

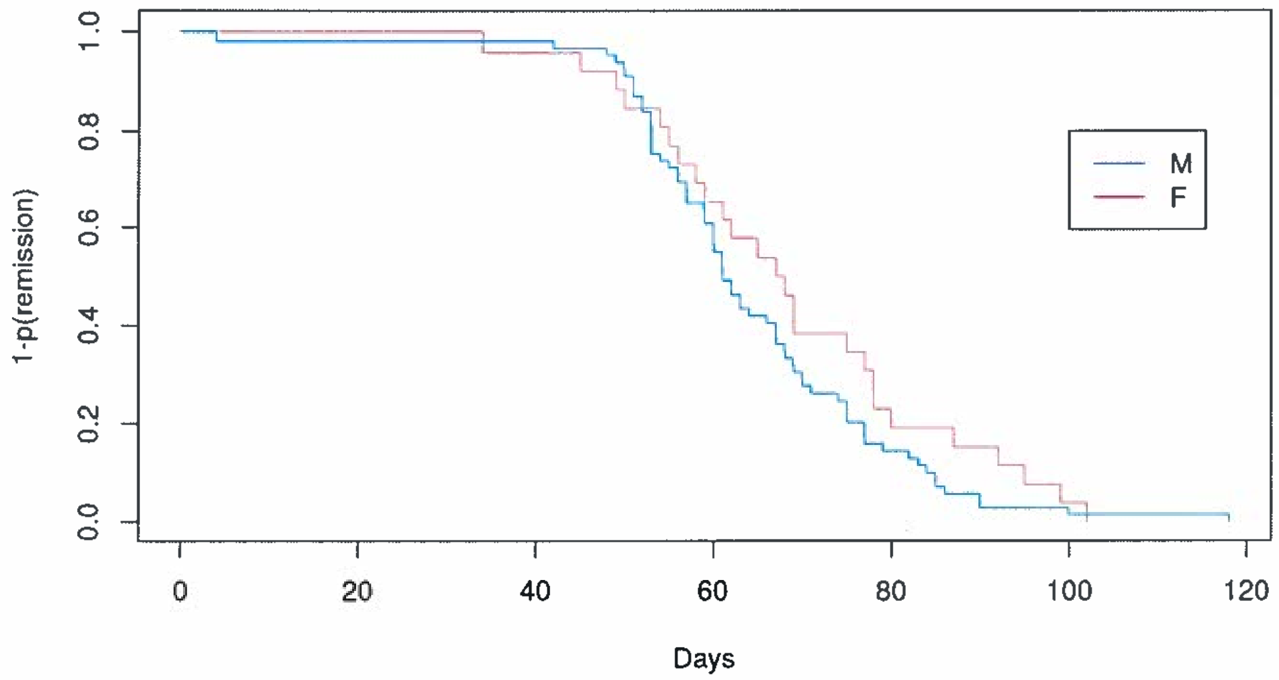
The survival time requires special treatment for a few reasons. Primarily,

- Survival times may be (and usually are) not normally distributed
- We need to properly address the issue of "censored" data – the incomplete datapoints due to subjects who either dropped from the study for unrelated reasons or did not have the event by the end of the study (so the true time to event is not known – we only know that it was longer than the time of last available assessment of the subject).

Survival analysis is well developed in R and we will start with an example and generate a "survival plot" in order to better illustrate the problem at hand (you may need to download the required libraries used in the code below).

Hide

```
library(survival) # Load survival analysis library
# copying old code here: extract days-to-remission:
ALL.pdat <- pData(ALL)
date.cr.chr <- as.character(ALL.pdat$date.cr)
diag.chr <- as.character(ALL.pdat$diagnosis)
date.cr.t <- strptime(date.cr.chr,"%m/%d/%Y")
diag.t <- strptime(diag.chr,"%m/%d/%Y")
d2r <- as.numeric(date.cr.t - diag.t)
# start the survival analysis:
d2r.ind <-as.numeric(!is.na(d2r)) # T at positions where d2r has data , False otherwise.
d2r.surv <- Surv(d2r,d2r.ind)
plot(survfit(d2r.surv~sex,data=pData(ALL)),
 col=c(2,4),xlab="Days",
 ylab="1-p(remission)")
legend(100,0.8,legend=c('M','F'),lwd=1,col=c('blue','red'))
```



In the code shown above we first reproduce several lines of code that we have already used on a few occasions in order to bring the days-to-remission data from ALL dataset into the R session. Next, we generate an auxiliary vector that has 'T' in the positions where time to event (days to remission) is known, and F otherwise. The functions from `survival` package are designed to work with a special object of class 'survival' that wraps around the actual data. Fortunately, a constructor function `Surv()` for the survival objects is provided, so in the next line we use this function in order to build the object. This function can take few different forms, depending on the type of data available, but in our case we simply pass the days to remission values as well as the auxiliary vector we just built.

The rationale for this seemingly excessive interface is that the survival data (times) are allowed to have different meanings: time value - for instance, "30" - can mean that event did occur exactly at that time (in which case the value at the corresponding position of the vector passed as the second argument to `Surv()`, 'event', should be set to `T`), or that all we know is that the event did not occur by that time but may have happened at some later time, i.e. event time is "30+". The latter case is referred to as right-censored data point, and it can be the result of either patient dropping out of the study (so we observed her up to the time point 30 and do not know what happened afterwards and when the event actually occurred) or termination of the whole study (e.g. the event did not occur in specific patient for the whole time we monitored our subjects). In our simple example we do not make much use of right-censored data points as we simply set the 'event' vector to 'FALSE' ("event did not occur") in positions were no time value is available at all (time to remission is `NA`).

With the survival object in hand, we can fit the survival model using the `survfit()` function. We fit the model against the patient's sex (a categorical variable; if we wanted to fit/plot a single survival curve for all patients, we would have to use the formula $d2r.surv \sim 1$) and plot the resulting fitted object right away in order to illustrate what the survival fit and survival curve actually are. Note that in our "survival" model the "event" is actually remission, not death, so the survival curve going down is a good thing: the height of the curve is the fraction of patients still not in remission. When the event is "death", the height of the survival curve is, correspondingly, the fraction of patients still alive, so the higher and the longer the curve runs, the better. As you can see in the figure generated by our $plot()$ command (shown below), the fit is stratified, according to the formula, into the curves for males and females; both genders seem to achieve remission in all patients by approx. day 100, and males seem to achieve remission slightly earlier.

Is the difference between the male and female curves in the survival plot we just generated significant? In other words, is it likely that males do achieve remission earlier than females, or the observed difference can be explained by random sampling and noise? The answer to this question is provided by Cox Proportional Hazards model (after Sir David Cox). Again, without going into much mathematical detail, we will only mention the ingredients of the hazards models:

- The "survival time" (time to event), T is viewed as a random variable. Indeed, we are observing the time to event in a randomly selected subject, so the event can happen pretty much anytime (and we do not know in advance when).
- The probability of the event to happen at different times can (and likely will) be different, of course, so our random variable $T$ is characterized by the cumulative distribution function $P(t) = Pr(T < t)$ (probability that event happens anytime before time $t$); or alternatively by the complementary "survival function" $S(t) = Pr(T > t) = 1 - P(t)$.
- The hazard at time t is defined as instantaneous risk (rate) of the event happening at time t:

$$h(t) = \lim_{\Delta t \to 0} \frac{Pr(t \leq T < +\Delta t | T \geq t)}{\Delta t}$$

Note that we use probability conditioned on survival up to the time t in the formula above: for the event to happen between t and $t + \Delta t$, it should not have happened by the time $t$!

- In the Cox model, the hazard function is represented as $logh_i(t) = \alpha(t) + \beta_1 x_{i1} + \ldots + \beta_k x_{ik}$ where $X_1, \ldots X_k$ are explanatory variables we are modeling the dependence of survival upon, and index i enumerates data points (observations, e.g. patients) – cf. "conventional" linear model $y_i = \alpha + \beta_1 x_{i1} + \ldots + \beta_k x_{ik}$.The strength (and weakness) of the Cox model lies in the fact that it is semi-parametric: it allows for the time dependent baseline hazard function $\alpha(t)$, which can be left unspecified (i.e. not parametrized). Since the baseline function is the same for all patients (and the parametrized part of the model describes departure from the baseline survival due to changes in the values of the explanatory variables $X_i$), we can still form and solve for hazard ratios: $\frac{h_i(t)}{h_j(t)} = \frac{exp(\beta_1 x_{i1}) + \ldots + \beta_k x_{ik}}{exp(\beta_1 x_{j1}) + \ldots + \beta_k x_{jk}}$, where the baseline function gets canceled out!

While the Cox method is strictly speaking inferior to, e.g. maximum likelihood estimate, the latter would require complete parametrization (including baseline), so in practice it would give reliable results only if we fully knew the right functional form of the hazard, including the baseline. In Cox model, the proportional hazards can be estimated (hence the name of the model) and the significance of the difference can be quantified; even the baseline hazard function can be actually estimated after the model is fitted.

Let us see how the model is implemented in R. All we need in order to fit the Cox proportional hazards model is the survival object (which we already built in the previous example). Continuing after the code shown above, we can run the following command directly:

Hide

```
coxph(Surv(d2r,as.numeric(!is.na(d2r)))~sex, data=pData(ALL))
```

```
## Call:
## coxph(formula = Surv(d2r, as.numeric(!is.na(d2r))) ~ sex, data = pData(ALL))
##
##         coef exp(coef) se(coef)     z     p
## sexM 0.2989    1.3483   0.2338 1.278 0.201
##
## Likelihood ratio test=1.7  on 1 df, p=0.192
## n= 95, number of events= 95
##    (33 observations deleted due to missingness)
```

In the code shown above we run proportional hazards model ( `coxph()` ) on a formula specifying dependence of the survival (interpreted as dependence of the hazard function in this case, according to the equations shown above) on the patient's sex. The output of the function shows that there is no significant difference ($p = 0.2$) between times to remission in males and females.