Lab 10 Part I
Linear regression and correlation analysis

Correlation analysis investigates the relationship between two continuous variables. If a significant relationship exists, you may want to predict the values of one variable from another, which is regression analysis. In linear regression analysis there are exact methods to fit parameters a and b of the function y=a+b*x, where a is the intercept with the y axis and b the slope of the regression line.

However, for more complex functions, curve fitting in R is based on trial and error. You have to know what the parameters of your function represent, and provide the R routines with start values for each parameter that you need to guess by looking at your data.

10.1 Dataset for linear correlation and regression analysis

Download or enter the following dataset, import to R, and check the import:

```
dat1 = read.csv("trees.csv")
head(dat1)
```

ID	DBH	VOL	AGE	DENSITY
2	11.5	1.09	23	0.55
3	5.5	0.52	24	0.74
4	11	1.05	27	0.56
5	7.6	0.71	23	0.71
6	10	0.95	22	0.63
7	8.4	0.78	29	0.63
9	8.4	0.77	21	0.64
12	9	0.87	27	0.6

10.2 Pearson correlations

Explore the relationship between the variables DBH, Vol, Age and Density and the significance of the relationships. This is the R code to calculate (1) a correlation coefficient, (2) test the significance of a correlation, (3) to create a matrix of correlation coefficients, (4) to create a matrix of coefficienc values for all pairs of variables. For the latter two, note that we first remove the ID column, which is not an independent or dependent variable:

```
cor(dat1$DBH,dat1$DENSITY)
cor.test(dat1$DBH, dat1$DENSITY)
cor(dat1$DBH, dat1$VOL)
cor.test(dat1$DBH, dat1$VOL)

dat2=dat1[,2:5]
cor(dat2)
cor(dat2)^2
```

If you don't want to carry out tests for a table of correlation coefficients, you can make use of a standard table, indicating significance of the correlation (r) as a function sample size (n). In our case with n=8 observations, all correlation coefficients above 0.71 are significant at α=0.05.

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Critical r	0.99	0.95	0.88	0.81	0.75	0.71	0.67	0.63	0.60	0.58	0.55	0.53	0.51	0.50	0.48

n	18	19	20	21	22	23	24	25	26	27	28	29	30	35	40
Critical r	0.47	0.46	0.44	0.43	0.42	0.41	0.40	0.40	0.39	0.38	0.37	0.37	0.36	0.33	0.31

n	45	50	60	70	80	90	100	125	150	175	200	250	300	500	1000
Critical r	0.29	0.28	0.25	0.24	0.22	0.21	0.20	0.18	0.16	0.15	0.14	0.12	0.11	0.09	0.06

10.3 Linear regression

If you want to use a significant linear relationship to make predictions, you need to derive an equation of the format y=b*x + a or "dependent variable" = "slope" * "independent variable" + "intercept". The statistical test for the significance of a regression function is a test of whether the slope of the regression is significantly different from zero. This is identical to the test of significant correlations above. You may also get an output that tests whether the intercept is significantly different from zero, which you would usually ignore unless this is of interest in the context of your analysis.

This is the code in R. The first line just gives you the formula, the second line the full range of statistics. Finally, we plot the regression line with ggplot2.

```
lm(dat1$VOL ~ dat1$DBH)
summary(lm(dat1$VOL ~ dat1$DBH))

library(ggplot2)

ggplot(dat1,aes(x=VOL,y=DBH))+
  geom_point()+
  stat_smooth(method="lm")
```

The gray ribbon indicates the 95% confidence interval. The command `se=F` can be used to turn it off.

10.4 Calculating r^2 manually (Optional, if you have time)

With the regression equation established, we can also calculate the r and r^2 value manually, just as we discussed in class. You need the differences between the individual observations and the overall mean of y (total variance in y), and the difference between the value predicted by the regression equation and the mean (variance explained). Then, we apply the standard formula for variances to get the ratio of both (r^2).

```
cor(dat1$DBH,dat1$VOL)# Note: r=0.9976258
lm(VOL~DBH,data=dat1) # Note: y=-0.0237+x*0.097
```

```
x=dat1$DBH
y=dat1$VOL
n=8

y_mean=mean(y)
y_pred=-0.0237+x*0.097

var_unexplained=sum((y-y_pred)^2)/(n-1)
var_total=sum((y-y_mean)^2)/(n-1)

r2=(var_total-var_unexplained)/var_total
r=sqrt(r2)
r                    # Note: compare to above
```

10.5 Non-parametric correlation analysis

Homogeneity of variances and normality in Y for a given X value can best be explored with residual plots. It's the same code that we used before in ANOVA (note that this typically looks better when your sample size is larger).

Residual plot, residuals, and histogram in R:

```
plot(lm(DENSITY~VOL,data=dat1))
res=residuals(lm(DENSITY~VOL,data=dat1))
hist(res)
```

If assumptions of normality and/or homogeneity of variances are violated, you can calculate correlation coefficients for non-parametric data. Both Kendall and Spearman rank correlations are very well regarded, robust test statistics for non-parametric correlations. However, keep in mind that they also assume linearity of the relationship.

```
cor(dat1$DBH,dat1$VOL, method = c("kendall"))
cor.test(dat1$DBH,dat1$VOL, method = c("kendall"))

cor(dat1$DBH,dat1$VOL, method = c("spearman"))
cor.test(dat1$DBH,dat1$VOL, method = c("spearman"))
```

10.6 Adjusting for multiple inference

If you want to draw general conclusions from a table of correlations (or in fact tables of any kind of statistical test) you need to make adjustments for multiple inferences in the same way that we discussed before for pairwise comparisons in ANOVA.

To illustrate the point to yourself, do the following experiment: The code below generates two random datasets and then carries out a correlation analysis. Obviously, the correlation between two random datasets should be zero. Execute the code multiple times (all three lines) and after a while you will find a significant relationship (about 1 out of 20 times if your alpha level is 0.05).

```
r1=rnorm(10)
r2=rnorm(10)
cor.test(r1,r2)
```

The more tests you execute, the more likely you will eventually make a type I error: rejecting the null hypothesis (of having no correlations) when it is true. If you are asking general questions based on multiple tests, you always have to protect yourself against this type I error inflation.

Here is a data table where we checked if growth is a function of climate variables. We reported the correlation between growth and monthly temperature with the corresponding p-value. The question is: "Is growth dependent on climate?" and the answer based on this table is "Yes, there are significant relationships with temperature in April, May, July, and August" at α=0.05.

Climate variable	**Correlation with growth**	**P value**
Temp Jan	0.03	0.4700
Temp Feb	0.24	0.2631
Temp Mar	0.38	0.1235
Temp Apr	0.66	0.0063
Temp May	0.57	0.0236
Temp Jun	0.46	0.1465
Temp Jul	0.86	0.0001
Temp Aug	0.81	0.0036
Temp Sep	0.62	0.0669
Temp Oct	0.43	0.1801
Temp Nov	0.46	0.1465
Temp Dec	0.07	0.4282

However, that's not quite right. You are drawing a general inference: "Climate influences growth", so you have to adjust for multiple inferences with the R code below, which returns adjusted p-values. What is your conclusion after adjustments?

```
table = read.csv("multiple_inference.csv")
p.adjust(table$Pvalue, method="holm", n=12)
```

There is a good paper written for biologists about Holm's adjustment method, and it also explains how you can do the adjustment manually (very easy). You can cite it in your own thesis/publications:

Rice, W.R. 1989. Analyzing tables of statistical tests. Evolution 43: 223–225.

Lab 10 Part II
Non-linear regression

Sometimes, your relationship between predictor and response variables may not be linear. The right type of non-linear model (exponential, power, logarithmic, polynomic) are usually conceptually determined based on biological considerations, e.g. exponential or restricted exponential growth (sigmoidal function), survival as a function of population size (hyperbolic or exponential), survival as a function of an environmental variable (often polynomic or Gauss functions), a biological process as a function of resource availability (Michaelis Menten functions). In such cases, you need different curve fitting techniques.

10.7 Sample dataset

Below is a dataset that you can use to practice fitting various non-linear functions. First, download or enter the data.

```
dat1 = read.csv("non_linear.csv")
head(dat1)
```

PV	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	2	0	0	50	0	0
1	1	2	2	18	32	33	4
2	3	4	4	24	25	49	12
3	7	5	15	29	21	64	33
4	15	11	42	33	18	69	65
5	20	16	72	35	14	68	90
6	29	22	83	37	13	62	73
7	40	35	85	40	10	53	40
8	52	51	86	39	10	32	11
9	65	77	85	40	9	0	1

Now make a series of plots of Y over your predictor variable PV, for example:

```
plot(dat1$Y1 ~ dat1$PV)
```

Alternatively, try using the ggplot2 package:

```
library(ggplot2) # Only needs to be run once

ggplot(dat1, aes(x = PV, y = Y1))+
  geom_point() + theme_bw()
```

Now repeat the plotting code with Y2, Y3, etc. Your decision what curve should be fitted should be based on visual inspection of the data and a theoretical expectation for the biological process: Y1 and Y2 is growth of a bacterial culture over time (i.e., logarithmic). Y3 is timber volume of a forest stand

over time (sigmoidal). Y4 is photosynthetic rate as a function of light (Michael Menten), Y5 is the nitrogen fixation rate of symbiotic bacteria as a function of fertilizer concentration (hyperbolic). Y6 and Y7 is survival as a function of temperature for two species (parabolic and Gaussian).

10.8 Sample formulas

Below are some sample formulas that we covered in class in the format that you can enter them into R or Excel. Remember that x and y are your independent and dependent variables, respectively. Meanwhile, a, b, and c are parameters. For some of the more complex formulas, the intercept has been removed, which you can add back in via an additional parameter (e.g. +d, if your intercept at x=0 is not y=0, but y=d instead.)

Hyperbolic: `y=a+b/(x+c)` # (approaching a and b)
Exponential: `y=a+b*c^x` # (approaching a)
Logarithmic: `y=a+b*x^c` # (approaching nothing)
Sigmoidal increase: `y=a/(1+(b^(-x+c)))`
Sigmoidal decrease: `y=a/(1+(b^(x-c)))`
Michaelis Menten increase: `y =(a*x)/(b+x)`
Michaelis Menten decrease: `y =a+(-a*x)/(b+x)` # (a version of half-life)
Parabolic upward: `y=a-b*(x-c)^2`
Parabolic downward: `y=a+b*(x-c)^2`
Gauss: `y=a*(b^((x-c)^2))`

10.9 Curve fitting

Curve fitting is a trial and error process, where the program starts with random values for your parameter, then the program increases or decreases that value and evaluates if the fit gets better or worse. If you have more than two parameters, the problem becomes very difficult. You can help the routine by providing a reasonable guess of the parameters as start value. Otherwise, R may not find a function that fits and end with an error message.

Try to fit a logarithmic function to Y1~PV. To make it a bit easier for R, the intercept parameter has been replaced with a number, as it seems to be zero. Then, the nls function should be able to figure it all out by itself. If things still don't work, specify start values as good as you can:

```
nls(dat1$Y1 ~ a*dat1$PV ^ b+0,data=dat1) # Probably will not work
nls(Y1~a* PV^b+0, start=list(a=1, b=2),data=dat1)
```

Now, we can fine tune the function by adding an intercept (`c`), because we now have perfect start values. This way, you can fit rather complicated functions in a step-wise fashion. To guess parameters, you need to look at the plots and visually estimate intercepts, maxima or asymptotes that are approached by the data. Note that the first request is too complex and results in an error, but the second will work:

```
nls(Y1~a*PV^b+c,data=dat1) # Won't work
nls(Y1~a*PV^b+c, start=list(a=0.88, b=1.95, c=0),data=dat1)
```

Finally, it is obviously essential to check visually if your curve fits the data. Using R base package functionality, the `curve` command let's you enter the parameters that you found with `nls` into the conceptual formula. (In the example below, that would be a logarithmic function: `Y1~a*PV^b` or generalized: `y=a*x^b)`. You have to skip `y=`, but you need to use `x` as a placeholder for your predictor variable. `add=T` simply adds the line to the previous plot command.

```
plot(dat1$Y1 ~ dat1$PV)
curve(0.8896*x^1.9539, add=T, lty=2)
```

If we want to visualize this in ggplot2, we cannot use the `curve` command but instead use the `stat_function()` command. Within the `stat_function()` command, we can specify the type of function with "fun":

```
ggplot(dat1, aes(x=PV, y=Y1)) +
  stat_function(fun=function(x)0.8896*x^1.9539, geom="line",
linetype="dashed",aes(colour="log")) +
  geom_point() + theme_bw() +
  scale_colour_manual(values = "grey")
```

Another alternative is to create your own function in R first and then call it in ggplot2:

```
mylogfun <- function(x) {
      0.8896*x^1.9539
      }

ggplot(dat1, aes(x=PV, y=Y1)) +
  stat_function(fun=mylogfun, geom="line",
linetype="dashed",aes(colour="log")) +
  geom_point() + theme_bw() +
  scale_colour_manual(values = "red")
```

Let's try one more example visualizing and fitting a sigmoidal function (Y3):

```
plot(dat1$Y3~dat1$PV)  # sigmoidal
nls(Y3~a/(1+(b^(-PV+c))),start=list(a=100, b=2, c=1),data=dat1)
plot(dat1$Y3~dat1$PV)
curve(85.9/(1+(4.892^(-x+4.002))),add=T,lty=2)
```

Using what we learned above, another option would be to plot our own function in ggplot2:

```
mysigfun <- function(x) {
  85.9/(1+(4.892^(-x+4.002)))
}

ggplot(dat1, aes(x=PV, y=Y3)) +
  stat_function(fun=mysigfun, geom="line",
linetype="dashed",aes(colour="log")) +
  geom_point() + theme_bw() +
  scale_colour_manual(values = "blue")
```

Try to find good visual fits for all pairs of variables (Y4~PV, Y5~PV, etc.).

10.10 Akaike's Information Criterion

It is best to pick the correct model based on a biological or physical understanding of the relationship. However, you can use Akaike's Information Criterion (AIC) to decide which model fits best. The criterion is useful because it can be calculated for any kind of model allowing comparisons across different modeling approaches and model fitting techniques.

AIC considers both the fit of the model and the numbers of parameters, where more parameters result in a "penalty". The model fit is measured as likelihood of the parameters being correct for the population based on the observed sample. The number of parameters is derived from the degrees of freedom that are left. AIC roughly equals the number of parameters minus the likelihood of the overall model, therefore *the smaller the AIC value, the better the model*.

Compare AICs for the following functions:

Simplified logarithmic:
```
AIC(nls(Y1~a*PV^b, start=list(a=1, b=2),data=dat1))
```
Logarithmic:
```
AIC(nls(Y1~a+b*PV^c, start=list(a=0, b=1, c=2),data=dat1))
```
Linear:
```
AIC(lm(dat1$Y1 ~ dat1$PV))
AIC(nls(Y1~a+b*PV, start=list(a=0, b=1),data=dat1))
```

If you were wondering which function to apply for your pairs of variables (Y2~PV, Y3~PV, Y4~PV, etc.), you can use the AIC criterion to find the best fit.