Power analysis

Contents

1 Introduction

To decide on whether a coefficient in a simple regression model is statistically significant, hypothesis test(s) are applied to make inference on that coefficient. Since the hypothesis tests will be applied on data gathered in empirical studies, these tests are performed after the data of the study has already been observed. In some cases, the hypothesis tests might show that the results are not statistically significant enough and therefore no conclusion regarding these observations can be drawn.

To understand what contributes to statistical significance, an empirical researcher might want to perform power analysis. Power analysis is a Monte Carlo method. In this case the process consists of applying regression in every replicate of the simulation, saving the corresponding p value and afterwards calculating the fraction of statistically significant p values.

The power of a test is the probability that when the alternative hypothesis is true, the null hypothesis will be correctly rejected (there is sufficient evidence that the null can be rejected). This depends on the elements of the study, such as the number of observations in the experiment, the effect size of the variable of interest or the variance of said variable.

Because power analysis is performed before the study has been conducted and requires a model to simulate data, some assumptions about the model need to be made (such as the probability distributions of the variables).

The following sections are dedicated to show what the power of a test depends on. The corresponding code is contained in the script file `PowerAnalysis.m` and can be found here on GitHub.

2 Baseline power analysis

As previously mentioned, the power of a test can be derived by simulating the data many times and count in how many simulations the results of interest are statistically significant. In the code we will first perform power analysis on a simple linear model. The model that we will consider is the following:

$$\boldsymbol{y} = \boldsymbol{X}\beta_1 + \boldsymbol{\varepsilon}$$

In the following code $\boldsymbol{X}$, $\boldsymbol{\varepsilon}$ and $\boldsymbol{y}$ all are $n \times 1$ column vectors. In the case of $\boldsymbol{X}$, all $x_{i1} \overset{\text{i.i.d.}}{\sim} N[0,1]$ (independent and identically normal distributed random variables) for all $i \in n$. $\boldsymbol{\varepsilon}$ contains the unexplained part of the model with $\varepsilon_i \overset{\text{i.i.d.}}{\sim} N[0,2]$ for all $i \in n$. Lastly, β_1 is set to 0.7. This model is the baseline model that we will consider for this subsection.

We now set $n = 80$ and let the code simulate the samples 500 times. In every simulation the code performs OLS and saves the p value of the estimated coefficient $\hat{\beta}_1$ in `B_p`. After the 500 simulations have been completed, we check in how many of the iterations the p value is below 0.05 (which is our preferred significance level). This fraction denotes the power of the test. To easily replicate the results, a seed has been set.

In all of the examples we will make use of the function file `exercisefunctionlss.m` to apply the linear regression. This function file can be found in the Appendix and on GitHub.

```
%% Calculating the power of a test
% Clear up the workspace
clear

% Setting the seed
rng(5)
```

```
7
8  % Setting up the base parameters
9  iter = 500;
10 n = 80;
11 B_1 = 0.7;
12 B_p = NaN(1, iter); % To store all the p values
13
14 % Multiple actions will be performed in the for loop below
15 %  1. Creating the linear model;
16 %  2. Use exercisefunctionlss() to get estimated coefficients (beta hat)
17 %     and the standard errors.
18 %  3. Calculate the corresponding p value store this value.
19 % Afterwards we determine the fraction of p values that have a value less
20 % than 0.05 (our preferred significance level). This fraction is the
21 % power of the test.
22
23 for i = 1:iter
24     % Create a sampling distribution for the OLS estimator
25     x_1 = random('Normal', 0, 1, n, 1);
26     u = random('Normal', 0, 2, n, 1);
27     y = x_1*B_1 + u;
28     LSS =  exercisefunctionlss(y, x_1);
29     % Test statistic
30     B_t = LSS.B_hat/LSS.B_hat_SEE;
31     B_p(i) = 2*cdf('T', -abs(B_t), n - LSS.K);
32 end
33
34 mean(B_p < 0.05) % Power of the test
```

The code determines that the power of the test is equal to 0.8440. This means that if the assumed model is indeed the true model, there is about a 84.4% chance that we will detect the true effect (with the current baseline model). Usually a power of at least 90% is favored (the preferred significance level). The current power of the test is therefore too low.

2.1 The effect of sample size

One way to increase the power is by increasing the sample size n. Since a sufficient sample size needs to be found through trial and error, we will make use of a for loop that iterates over a set of sample sizes.

```
1  %% Increasing the sample size to find a better power
2  % We have now calculated the power in the case of 80 samples and an
3  % effect size of 0.7. This power is not satisfying as we want to have
4  % a power of at least 90%.
5
6  % A satisfying power can be found through trial and error. It is therefore
7  % efficient to create a loop that calculates the power for multiple values
8  % of the parameter of interest (e.g. the sample size or effect size).
9  % Let's first focus on the sample size. We will create a for loop that
10 % stores the power for specific sample sizes.
11
12 clear
13 rng(5)
14 iter = 500;
```

```
B_1 = 0.7;
B_p = NaN(1, iter);

n_samples = (120:20:220); % The sample sizes that we will test for

N_and_Power = table(n_samples', NaN(length(n_samples), 1), ...
    'VariableNames', ["Sample size", "Power"]);

for j = 1:length(n_samples)
    for i = 1:iter
        x_1 = random('Normal', 0, 1, n_samples(j), 1);
        u = random('Normal', 0, 2, n_samples(j), 1);
        y = x_1*B_1 + u;
        LSS = exercisefunctionlss(y, x_1);
        B_t = LSS.B_hat/LSS.B_hat_SEE;
        B_p(i) = 2*cdf('T', -abs(B_t), n_samples(j) - LSS.K);
    end
    N_and_Power.Power(j) = mean(B_p < 0.05);
end

N_and_Power;

figure
plot(N_and_Power, "Sample size", "Power");
title("How sample size influences the power")
yline(0.9, '--');

% The output tells us that a sample size of close to 100 samples gives
% sufficient power.
```

After performing the calculations, we summarize the result in Figure 1. We have also plotted a horizontal dashed line in the figure to indicate our minimal required power.

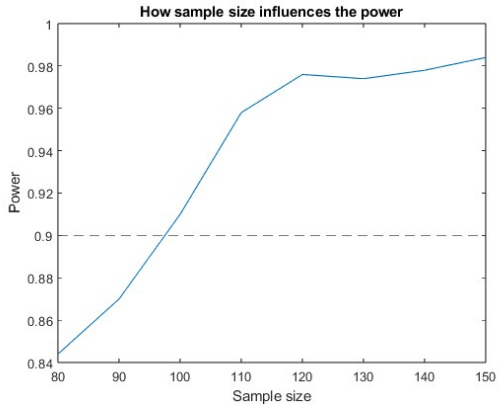

Figure 1: The effect of sample size on the power

Since the goal is to find a sample size that gives a power of at least 90%, a sample size close to 100 observations results in sufficient power.

2.2 The effect of size of the true effect

It is not always in the discretion of the empirical researcher to collect more data (to get a larger sample size). Besides sample size, the effect size is also a determinant of the power. Here we will demonstrate what happens with the power when the true effect size is small and what size of β_1 will give a sufficient power.

```
%% Effect size as the determinant of the power
% Now we set the number of samples back to 80 and instead calculate the
% power for different effect sizes.

clear
rng(5)
iter = 500;
n = 80;
B_p = NaN(1, iter);

effectsizes = (0.1:0.1:0.9); % The effect sizes that we will test for

Effect_and_Power = table(effectsizes', NaN(length(effectsizes), 1), ...
    'VariableNames', ["Effect", "Power"]);

for j = 1:length(effectsizes)
    for i = 1:iter
        x_1 = random('Normal', 0, 1, n, 1);
        u = random('Normal', 0, 2, n, 1);
        y = x_1*effectsizes(j) + u;
        LSS = exercisefunctionlss(y, x_1);
        B_t = LSS.B_hat/LSS.B_hat_SEE;
        B_p(i) = 2*cdf('T', -abs(B_t), n - LSS.K);
    end
    Effect_and_Power.Power(j) = mean(B_p < 0.05);
end

Effect_and_Power;

figure
plot(Effect_and_Power, "Effect", "Power")
title("How effect size influences the power")
yline(0.9, '--');

% This graph shows that the minimum detectable effect size of the
% coefficient in the case of 80 samples is a little more than 0.7.
```

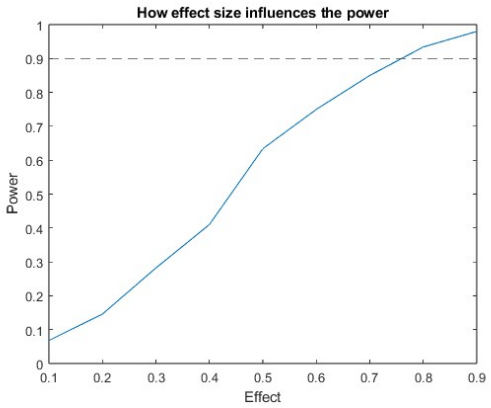

Figure 2: The effect of size of the true effect

Figure 2 shows that observing true effect sizes a little larger than 0.7 will have enough statistical power. This means that if the true effect size is 0.7 or less, the detection of this effect size will not be statistically significant (with the current model and number of observations).

2.3 The effect of variance in observed values of the predictor

Let us now set the sample and effect size return to their original values from the baseline model. In the code below we analyze what happens to the power of the test when the variance of the predictor increases.

```
%% The effect of variance on the power
% Let's set the effect size back to its original values and now focus on
% the variance of x_1.

clear
rng(5)
iter = 500;
n = 80;
B_1 = 0.7;
B_p = NaN(1, iter);

varianceset = 0.6:0.2:1.4; % The set of variances that we will test for

Variance_and_Power = table(varianceset', NaN(length(varianceset), 1), ...
    'VariableNames', ["Variance", "Power"]);
for j = 1:length(varianceset)
    for i = 1:iter
    x_1 = random('Normal', 0, sqrt(varianceset(j)), n, 1);
    u = random('Normal', 0, 2, n, 1);
    y = x_1*B_1 + u;
    LSS = exercisefunctionlss(y, x_1);
    B_t = LSS.B_hat/LSS.B_hat_SEE;
    B_p(i) = 2*cdf('T', -abs(B_t), n - LSS.K);
    end
    Variance_and_Power.Power(j) = mean(B_p < 0.05);
end

```

```
Variance_and_Power;

figure
plot(Variance_and_Power, "Variance", "Power")
title("How variance influences the power")
yline(0.9, '--');

% The above table shows us that as the variance of x_1 increases, the
% power of the test also increases; the (total) variance of y becomes
% 'relatively more explained' by the variance of x_1. In this example,
% the minimum variance required for a satisfying power is inbetween 1.1
% and 1.2.
```

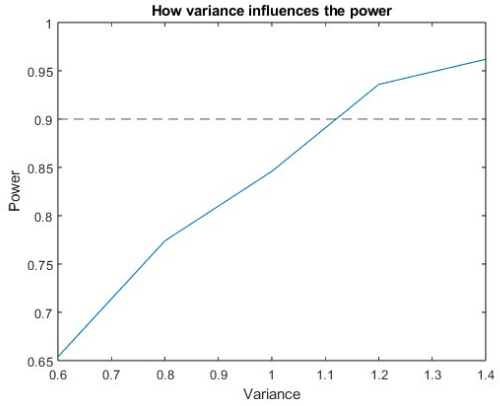

Figure 3: The effect of variance on the power

From Figure 3 it can be observed that as the variance of the observed values of the predictor increases, the power of the test also increases. As is mentioned in the comments in the code, this is because now a larger fraction of the variance in $\mathbf{y}$ is explained by the variance of the observed variable.

3 Change of model

We now change the model slightly to allow us to find out how correlation between two predictors affect the statistical power. The dimensions of vectors $\mathbf{y}$ and $\boldsymbol{\varepsilon}$ remain the same, but $\mathbf{X}$ will contain an extra column that represents $\mathbf{x}_2$. We will now make use of $\boldsymbol{\beta}$:

$$\boldsymbol{\beta} = \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} 0.7 \\ 0.75 \end{pmatrix}$$

Since the goal is to study the correlation, we need to have that:

$$(\mathbf{x}_1, \mathbf{x}_2)^T \sim N[\mathbf{0}, \boldsymbol{\Sigma}]$$

$\mathbf{x}_1$ and $\mathbf{x}_2$ are bivariate normal distributed where $\boldsymbol{\Sigma}$ is the corresponding covariance matrix. By letting σ_1 and σ_2 denote the standard deviation of $\mathbf{x}_1$ and $\mathbf{x}_2$ respectively and ρ the correlation between these variables, the following setup is used:

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2\rho \\ \sigma_1\sigma_2\rho & \sigma_2^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

With these values for $\mathbf{\Sigma}$, there is zero correlation between the predictors and therefore both predictors are independent and identically $N[0, 1]$ distributed.

```
%% Changing the model
% To show how correlation between two variables affects the power, we need
% to make some small changes to our model. x_2 will be added to our model
% and we will let x_1 and x_2 be bivariate normal distributed. To show
% that the power of the test is not much different than the power in the
% first example, we will use an identity matrix for the variance-
% covariance matrix for the bivariate normal distribution (as this means
% that there is no correlation between x_1 and x_2).

clear
rng(5)
iter = 500;
n = 80;
B_1 = 0.7;
B_2 = 0.75;
B_true = [B_1; B_2];
B_p = NaN(1, iter);

for i = 1:iter
    x_1_x_2_bivariate = mvnrnd([0 0], [1 0; 0 1], n);
    x_1 = x_1_x_2_bivariate(:,1);
    x_2 = x_1_x_2_bivariate(:,2);
    X = [x_1 x_2];
    u = random('Normal', 0, 2, n, 1);
    y = X*B_true + u;
    LSS = exercisefunctionlss(y, X);
    B_t = LSS.B_hat(1)/LSS.B_hat_SEE(1);
    B_p(i) = 2*cdf('T', -abs(B_t), n - LSS.K);
end

mean(B_p < 0.05)

% It can be observed that the power of the test is almost equal to the
% power in the first example.
```

The statistical power is estimated to be 0.8720, which is slightly larger than the power in the first example since the fraction of variance of $\mathbf{X}$ in the variance of $\mathbf{y}$ has increased with the introduction of $\mathbf{x}_2$.

3.1 The effect of correlation between random variables

The goal of this subsection is to find out how the correlation between the two predictors changes the power of the test. This will be done by gradually increasing the value of the correlation coefficient.

```
%% Correlation
% To allow for more ease in changing the bivariate normal distribution, we
% will use sigma_1 and sigma_2 to denote the standard deviations of x_1
% and x_2 respectively in the case that these variables were uncorrelated
% and normal distributed (with mean zero and standard deviations sigma_1
% and sigma_2 respectively).

clear
rng(5)
iter = 500;
n = 80;
B_1 = 0.7;
B_2 = 0.75;
B_true = [B_1; B_2];
sigma_1 = 1;
sigma_2 = 1;
B_p = NaN(1, iter);

correlationset = [0:0.1:0.9 0.99];

Correlation_and_Power = table(correlationset', ...
    NaN(length(correlationset), 1), 'VariableNames', ["Correlation", ...
    "Power"]);
for j = 1:length(correlationset)
    for i = 1:iter
    x_1_x_2_bivariate = mvnrnd([0 0], [sigma_1^2 ...
        correlationset(j)*sigma_1*sigma_2; ...
        correlationset(j)*sigma_1*sigma_2 sigma_2^2], n);
    x_1 = x_1_x_2_bivariate(:,1);
    x_2 = x_1_x_2_bivariate(:,2);
    X = [x_1 x_2];
    u = random('Normal', 0, 2, n, 1);
    y = X*B_true + u;
    LSS = exercisefunctionlss(y, X);
    B_t = LSS.B_hat(1)/LSS.B_hat_SEE(1);
    B_p(i) = 2*cdf('T', -abs(B_t), n - LSS.K);
    end
    Correlation_and_Power.Power(j) = mean(B_p < 0.05);
end

Correlation_and_Power;

figure
plot(Correlation_and_Power, "Correlation", "Power");
title("How correlation influences the power");
ylim([0 1]);
yline(0.9, '--');

% The table shows that when the correlation between x_1 and x_2 increases,
% the power of the test decreases. To improve the power, the same 'tricks'
% can be used again:
%   - Increasing the sample size;
%   - Increasing the effect size of the random variable(s).
```

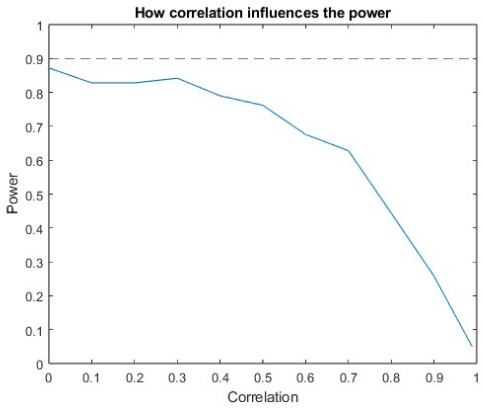

Figure 4: The effect of correlation on the power

The above figure shows that as the value of the correlation coefficient increases, the statistical power decreases and that this decrease in power speeds up. This result gives insight on multiple regression: highly correlated predictors (multicollinearity) can lead to the power of the test being low.

4 Significance level

Another way to increase the power of the test, albeit not that useful, is changing the preferred level of statistical significance. In all of the code, we only deem a replicate to be 'successful' when the p value is less than 0.05. Identifying simulations to be 'successful' for a larger p value will result in the total fraction of satisfying results to increase and therefore producing a larger statistical power.

5 Appendix: LSS Function

```
function LSS = exercisefunctionlss(y,X)
%% Number of observations and column dimension of X
LSS.N                  = length(y);
LSS.K                  = size(X,2);
%% Coefficient estimates, predictions, residuals
LSS.B_hat              = inv(X'*X)*X'*y; % Or (X'*X)\X'*y.
LSS.y_hat              = X*LSS.B_hat;
LSS.u_hat              = y-LSS.y_hat;
%% Residual sum of squares
LSS.RSS                = LSS.u_hat'*LSS.u_hat;
%% The estimator of the variance of the regression error
LSS.sigma_hat_squared  = LSS.RSS/(LSS.N-LSS.K);
LSS.sigma_hat          = sqrt(LSS.sigma_hat_squared);
%% The variance-covariance estimator of the OLS estimator
LSS.B_hat_VCE          = LSS.sigma_hat_squared.*inv(X'*X);
LSS.B_hat_SEE          = sqrt(diag(LSS.B_hat_VCE));
end
```